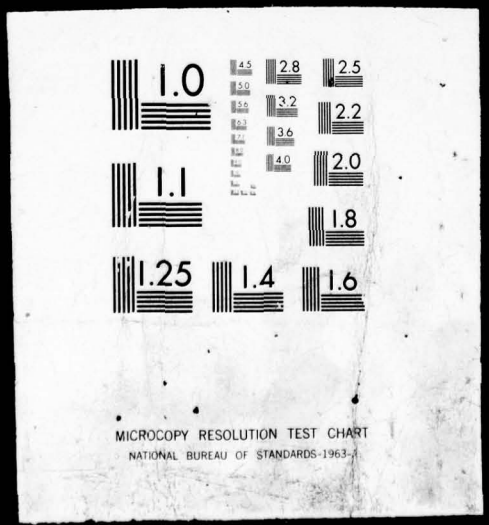
AD-A084 215 VIRGINIA POLYTECHNIC INST AND STATE UNIV WASHINGTON --ETC F/G 9/2
DESIGN FOR A CMS SIMULA FILE SYSTEM WITH 4 CHARACTER SETS.(U)
NOV 79 R J ORGASS AFOSR-79-0021
UNCLASSIFIED VPI/SU-TM-79-8A AFOSR-TR-80-0337 NL

1 OF 2

AD

AO 84 215



EXTENSION DIVISION

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
GRADUATE PROGRAM IN NORTHERN VIRGINIA

P. O. Box 17186
Washington, D. C. 20041
(703) 471-4600

DESIGN FOR A CMS SIMULA* FILE SYSTEM WITH 4 CHARACTER SETS†§

Richard J. Orgass

Technical Memorandum No. 79-8a

November 5, 1979

ABSTRACT

Specifications for the SIMULA file classes which are proper extensions of the Common Base Definition are given. The following assumptions are used in the design. (1) A file is named only by a file specification (an extension of a CMS fileid); simply mentioning the name of a file causes it to exist. (2) A file is a sequence of characters divided into records by <newline>. This is implemented using standard CMS files in a way that is transparent to the program. (3) A file may be written in one of four character sets: EBCDIC, key-paired APL, bit-paired APL and the IBM APL print train set. (4) Independent of the file character set, the file may be read in EBCDIC or key-paired APL with appropriate character translation provided by the file system.

---

* SIMULA is a registered trademark of the Norwegian Computing Center, Oslo, Norway.

† Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, under Grant No. AFOSR-79-0021. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon.

§ The information in this document is subject to change without notice. The author, Virginia Polytechnic Institute and State University, the Commonwealth of Virginia and the United States Government assume no responsibility for errors that may be present in this document or in the program described here.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (19) (18) AFOSR-TR-80-0337	2. GOVT ACCESSION NO. AD-A084215
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)* (6) DESIGN FOR A CMS SIMULA FILE SYSTEM WITH 4 CHARACTER SETS	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Richard J. Orgass	8. CONTRACT OR GRANT NUMBER(s) (15) AFOSR-79-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Virginia Polytechnic Inst. & State University, Department of Computer Science, Washington, DC 20041	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F (16) 2304/A2 (17)
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM, Bolling AFB, Washington, DC 20332	12. REPORT DATE (11) November 1979
	13. NUMBER OF PAGES: 112
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* (9) Technical memo,	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

(12) 31

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

(14) VPI/SU-TM-79-8a

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Specifications for the SIMULA file classes which are proper extensions of the Common Base Definition are given. The following assumptions are used in the design. (1) A file is named only by a file specification (an extension of a CMS fileid); simply mentioning the name of a file causes it to exist. (2) A file is a sequence of characters divided into records by new line . This is implemented using standard CMS files in a way that is transparent to the program. (3) A file may be written in one of four character sets: EBCDIC, key-paired APL, bit-paired APL and the IBM APL print train set. (4) Independent

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

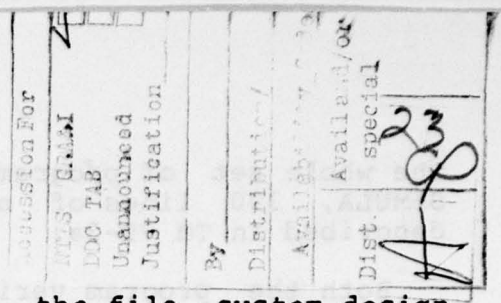
411731

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. Abstract cont.

of the file character set, the file may be read in EBCDIC or key-paired APL with appropriate character translation provided by the file system.



1. Introduction

Perhaps the easiest way to motivate the file system design described here is to explain how the author was motivated to undertake a substantial program development effort that is quite orthagonal to his research interests.

I brought several large strongly interactive DEC-10 SIMULA programs to Virginia Tech for use in VM/CMS. After resolving problems related to character set translation and a few bugs in the IBM SIMULA compiler, the programs behaved more or less the same in VM and in TOPS-10. There was, however, one important problem area that remained: terminal dialog and file management. After three months of work on ad hoc program modifications to solve problems and on learning some of the obscure details of CMS, I decided that a general solution to these problems was the only reasonable way to proceed.

My first approach to this problem was to design and implement a SIMULA class DIALOG which contains procedures to efficiently rewrite the terminal transaction part of my programs. The design of all of my programs assumed that a file can be opened at run time simply by mentioning the name of the file. CMS SIMULA, as distributed, requires that files be given DD names using filedef commands before program execution begins. It became very obvious that it was completly unreasonable to redesign my programs to reference files in this way. Before beginning the verification of programs, one simply doesn't know all the file names that will be used. Therefore, this first version of DIALOG had some rudimentary procedures for naming files at run time using only CMS fileids. This version of DIALOG is described in TM 79-3.

It very quickly became obvious that this version of DIALOG was inadequate with respect to dynamic file naming. There were simply too many things that could go wrong resulting in a termination of execution and substantial loss of work. R. E. Porter then made major changes in DIALOG to provide adequate security against errors in fileids and other input/output errors. This new version proved quite useful and solved many problems. There were still a few details of CMS file format that intruded occasionally but they could be solved with some special code and/or the copy command.

At this point, one important problem appeared to remain: It was not possible to write terminal output without a trailing <cr><lf>. In addition, two calls to Outimage (which caused redundant blank lines in the terminal transcript) were required to have prompts precede the read for a response. To solve these problems, R. D. Johnson wrote two assembly procedures to bypass the SIMULA system output code and these procedures were imbedded in class Outfile. This quite adequately solved the problem but

the whole set of programs were now quite baroque (2500 lines of SIMULA, 300 lines of assembler). This version of DIALOG is described in TM 79-3a.

Both the program verification system and its associated APL implementation must interact with both ASCII and APL character set devices and there are three APL character set codes: key-paired APL, bit-paired APL and the IBM APL print train character set. Character set management and translation is an appropriate part of the input/output system and extensions to DIALOG to provide this capability were designed and experimental implementations were studied. It became quite obvious that programs built on top of such a DIALOG would be inefficient and lack reliability because such a DIALOG would be extended far beyond what was anticipated in the original design.

Therefore, a comprehensive design of a file system that is a proper extension of the SIMULA Common Base Definition was undertaken. The view of files adopted in this design is quite different from the standard CMS view and should prove to be much easier to use.

The first assumption is that a file is named only by a file specification. Moreover simply writing to a file after giving its file specification is sufficient to cause the file to exist. Running programs may dynamically reference both input and output files using only the file specification. The concept of a DD name does not exist. If the specification of a file changes between executions of a program, then the file specification is read as data.

The second assumption is that a file may be associated with any input or output device and that from the vantage point of a running program there is no difference between devices except for the text of the file specification. The various file formats of CMS are of no concern to the program.

The third assumption is that a file is a sequence of characters divided into records by <newline>. One does not distinguish between fixed and variable length records, etc., etc. Files are read or written one record or line at a time without distinction.

It is assumed that the data in a file may be written in one of four character sets: EBCDIC, key-paired APL, bit-paired APL and APL print train code. Furthermore, a program may interpret characters coming from a file or written to a file as either a sequence of EBCDIC characters or as a sequence of key-paired APL characters. It is the responsibility of the file system to provide the appropriate translation.

A design requirement is that the files actually read and written must be standard CMS files so that programs written using this file system can communicate with other programs by way of the CMS file system.

The file specification used here is an extension of the CMS fileid to include device names and character set specifications. The file system uniformly communicates with disk files, the terminal and the virtual reader, printer and punch. Magnetic tapes are not included because they are not available to VM users at Virginia Tech.

2. Files

In a running program, a file is viewed as a sequence of characters divided into records or lines by <new line> characters. Files are read or written one record at a time and individual records may be blocked into a standard length by the input/output procedures to simplify programming. Except for the file specification, a running program need not distinguish between different input/output devices or the character set of the device. This means that the programs that read from or write to a peripheral must present a uniform interface to user programs.

The file system described here recognizes disk files and four peripheral devices:

Device	File Specification
terminal	tty: or con:
virtual printer	lpt: or prt:
virtual punch	pun:
virtual reader	rdr:

There are two formats for file specifications for disk files and these two formats define different actions on the part of the input/output system.

The more commonly used file specification for a disk file is the usual CMS fileid which is of the form:

<fn>[<ft>[<fm>]]

The blanks in the file specification may be replaced by periods (.). If <ft> and/or <fm> is omitted, default values are provided. These default values are different for different kinds of files.

For Infiles, if <ft> is omitted the <ft> is set to DATA. After the <fn> and <ft> are completed, the <fm> is provided as follows. All of the accessable disks of the active job are searched for a file with the given <fn> <ft>. The first file that matches this partial specification in the standard search order and which is available for reading is selected.

For Outfiles, if <ft> is omitted the string LOG is provided as <ft>. After the <fn> and <ft> is specified, the file mode is provided as follows. Each disk that is available for writing is searched in the standard search order. The following is a sketch of the algorithm:

```
IF "there is file <fn> <ft> on the disk"
    THEN "replace the existing file"
    ELSE IF "there is space on the disk"
            THEN "create a new file
                  <fn> <ft> on the disk"
            ELSE "go on to the next disk"
```

Defaults for Printfiles are completed in the same way as Outfiles except that the default <ft> is the string LISTING.

A single period is acceptable in place of the blanks that usually separate the components of a CMS fileid. In addition, the string <fn>..<fm> specifies the <fn> and <fm> while selecting the default <ft>.

The second form of the file specification is of the form

```
fil:<string>
```

where <string> is any EBCDIC character string. This file specification indicates that the actual file specification is to be read from the terminal when a file object is created; see Section 4, below.

A complete file specification includes a character set specification which is described in Section 3, below. The syntax of a file specification is:

```
tty: | con: | lpt: | prt: | pun: | rdr: |
   fil:<string> |
  <fn>[{{ |.}<ft>[{ |.}<fm>]|..<fm>}][<character set spec>]
```

3. Character Sets

This file system is designed for use with APL and with the usual character set. It is assumed that each peripheral device may have any one of four different character sets associated with it. The four character sets are:

```
EBCDIC
KEY
BIT
APL
```

The character set EBCDIC refers to the EBCDIC equivalent of ASCII as defined by the translate tables used with the Memorex

1380 at the Virginia Tech Computing Center. This table is attached as Appendix A.

The character set KEY refers to the typewriter pairing that maps the APL character set into ASCII as defined in the STAPL convention. After mapping the APL characters into ASCII, the ASCII characters are mapped into EBCDIC as above. The STAPL Convention is attached as Appendix B.

The character set BIT refers to the bit pairing that maps the APL character set into ASCII as defined in the STAPL Convention. After mapping the APL characters into ASCII, the ASCII characters are mapped into EBCDIC as above.

The character set APL refers to the character code used to map EBCDIC codes into printable characters using the APL print train for IBM printers.

The actual contents of a file may be in a specific character set but the program may wish to read or write the file using a different character set. For example, an APL interpreter might well wish to read an ASCII file as though it were a key-paired APL file but using a translation that permits input or output for devices that lack the APL character set. Thus, the specification of the character set associated with a file must define both the character set used in the file and the character set that the program is interpreting. However, for reasons of simplicity, it is desirable to have only one APL character set within a program. The key-paired APL character set has been selected.

There are two forms of a character set specification, a short form and a long form.

A short character set specification is of the form:

/<file char set>.<prog char set>

<file char set> is the character set in which the file is read or written and <prog char set> is the character set in which the program wishes to interpret the file.

Not all combinations of character sets are sufficiently useful to merit implementation. For input files, the following character set specifications are implemented:

/EBCDIC.EBCDIC
/EBCDIC.KEY
/BIT.KEY
/APL.KEY

For output files, the following character set specifications are implemented:

```
/EBCDIC.EBCDIC
/KEY.KEY
/BIT.KEY
/APL.KEY
/EBCDIC.KEY
/KEY.EBCDIC
/BIT.EBCDIC
```

If the character set specification is omitted, the default for both the file and program character set is EBCDIC. If the character set specification is /EBCDIC or /KEY then both the file and program character set are set to EBCDIC or KEY.

The short form of the character set specification is difficult to read and remember and this may cause program errors. The long form of the character set specification provides mnemonics to make it easier to remember the format. For input files, the following eight long form character set specifications are acceptable:

```
/FILE EBCDIC PROG EBCDIC     /PROG EBCDIC FILE EBCDIC
/FILE KEY PROG KEY           /PROG KEY FILE KEY
/FILE BIT PROG KEY           /PROG KEY FILE BIT
/FILE APL PROG KEY           /PROG KEY FILE APL
```

For output files, the following fourteen long form character set specifications are acceptable:

```
/FILE EBCDIC PROG EBCDIC     /PROG EBCDIC FILE EBCDIC
/FILE KEY PROG KEY           /PROG KEY FILE KEY
/FILE BIT PROG KEY           /PROG KEY FILE BIT
/FILE APL PROG KEY           /PROG KEY FILE APL
/FILE EBCDIC PROG KEY        /PROG KEY FILE EBCDIC
/FILE KEY PROG EBCDIC        /PROG EBCDIC FILE KEY
/FILE BIT PROG EBCDIC        /PROG EBCDIC FILE BIT
```

Since the short form of the character set specification is much simpler when only one character set is used, there are no defaults for the long form.

The specification of the character set translations is completed when each of the eleven translations is specified.

If the file character set and the program character set are identical, then no translation is performed on input or output. This applies to the specifications /EBCDIC.EBCDIC and /KEY.KEY.

If the character set of an input file is EBDIC and the program character set is KEY then the following translation is performed.

(1) Lower case EBCDIC letters are mapped into APL letters.

(2) Upper case EBCDIC letters are mapped into underlined APL letters.

(3) Escape sequences of the form .xx are mapped into (one or three) key paired characters as defined in the APLSF and APL.MS implementations (Appendix C).

(4) Those EBCDIC graphics which are mapped into APL characters by the translation in Appendix C are translated into key-paired characters as specified.

(5) Those EBCDIC graphics which are not mapped into APL characters in accord with (4) are mapped into key-paired characters in accord with the inverse of the mapping defined by the array key_paired shown in Appendix D.

If the character set of an input file is BIT and the program character set is KEY then the bit-paired characters are mapped into key-paired characters as defined by Figures 4 and 5 of Appendix B.

If the character set of an input file is APL and the program character set is KEY then the input characters are mapped into key-paired APL so as to preserve the appearance of the line. Strikeovers are, of course, mapped into three characters. A copy of the character set for the APL train must be secured from the Computing Center.

There are five more translations available for output files:

For output files, if the program character set is KEY and the file character set is BIT, then character translation as defined by Figures 4 and 5 of Appendix B is performed.

For output files, if the program character set is KEY and the file character set is EBCDIC, then character translation that is performed is the inverse of the mapping defined for input with character set specification /FILE EBCDIC PROG KEY.

For output files, if the program character set is KEY and the file character set is APL, then the character translation that is performed preserves the appearance of the line for the APL print train. The character codes for the print train must be secured from the Computing Center.

For output files, if the program character set is EBCDIC and the file character set is KEY, then the output is translated into key-paired APL using the mapping defined for input files when the file character set is EBCDIC and the program character set is KEY.

For output files, if the program character set is EBCDIC and the file character set is BIT, then the output is translated into key-paired APL using the mapping defined for input files when the file character set is EBCDIC and the program character set is KEY. Next, the key-paired APL is translated into bit-paired APL using Figures 4 and 5 of Appendix B.

In the above discussion, character set translation is defined in terms of several mappings. The actual program performs each translation directly without a sequence of translations.

The APL character set may not be used with the terminal. If the specified file character set for a file connected to the terminal is APL, an error message is written to the terminal (a ? message) and a corrected file character set is read or execution is terminated.

4. Common Specifications

The declarations of classes Infile, Outfile and Printfile used in this file system satisfy the definitions given in the SIMULA Common Base Definition with certain extensions. Those extensions which apply to all three file classes are described in this section and extensions of the individual file classes are described in subsequent sections.

The parameter of a file class is a file specification as defined above. When a class instance is created, the file specification is processed as follows:

(1) The character set specification is separated from the file specification and translated into upper case under the assumption that it is an EBCDIC string. This text is analyzed and used to set the values of the variables fil_set and prog_set as follows:

Character Set	Value
EBCDIC	0
KEY	1
BIT	2
APL	3

If there is an error in the character set specification, the entire file specification together with a description of the error is written to the terminal and the user is asked to provide a correct character set specification.

(2) The remainder of the file specification is translated into upper case EBCDIC in accord with the program character set.

(3) The missing components of the fileid are provided in accord with the defaults defined in Section 2.

(4) The file specification is checked for syntax errors. If there are syntax errors, a message is written to the terminal describing the error (see get_dd_input and get_dd_output in DIALOG) and the user is given an opportunity to provide a correct specification.

(5) The directory is searched for the appropriate entry as defined in Section 2. If the file cannot be found a corrective message is written to the terminal and the user is asked to provide a corrected file specification (but not a character set specification). At this point, the user may enter CMS subset to look for the appropriate file by selecting the default answer CMS:. Upon returning from the CMS subset, the user is again asked to name the file.

(6) The value of f_spec is set to the$CMS file specification as translated; the character set specification is removed.

If the file specification is of the form FIL:<string>, then the message:

% Please enter the file specification for file <string>:

is printed on the terminal and the user response is interpreted as above.

All of the usual procedure attributes of file objects will first check to see if the file is open. If the file is closed, the user will be given an opportunity to open the file. See the code for the attributes of class out_file in file OUTFILE SIMULA for examples.

If the procedure Open is called when the file is already open, a message of the form

[% File '<fileid>' is already open.]

is printed on the terminal and execution continues.

If the procedure Close is called when the file is already closed, a message of the form

[% File '<fileid>' is already closed.]

is printed on the terminal and execution continues.

All file objects will have integer procedure attributes program_set and file_set whose return value is the character set number of the program and file, respectively, associated with the file object.

Each file class has a text procedure attribute file_spec whose return value is the fileid or device specification of the file associated with the file object. [Most precisely, the file specification without the character set specification.]

Each instance of a file class is to have two integer stacks implemented as linked lists using SIMULA classes which will be used to store program and file character set numbers.

When the procedure enter_ascii is executed, the following occurs:

(1) The current program character set number is pushed onto the program character set stack.

(2) If the file is an Infile, Image is set to Blanks(Image.Length). If the file is an Outfile or Printfile, Outimage is executed if Image.Strip =/= NOTEXT.

(3) The program character set number is set to 0.

When the procedure enter_key is executed, the following occurs:

(1) The current program character set number is pushed onto the program character set stack.

(2) If the file is an Infile, Image is set to Blanks(Image.Length). If the file is an Outfile or Printfile, Outimage is executed if Image.Strip =/= NOTEXT.

(3) The program character set number is set to 1.

When the procedure restore is executed, the following occurs:

(1) If the file is an Infile, Image is set to Blanks(Image.Length). If the file is an Outfile or Printfile, Outimage is executed if Image.Strip =/= NOTEXT.

(2) The current program character set is set to the character set number that is on the top of the stack and the stack is popped.

The procedures set_ascii, set_bit, set_key, set_apl and reset are used to change the file character set.

The three set procedures perform the following actions:

(1) If the file is an Outfile or Printfile, then if Image.Strip NE NOTEXT, Outimage is called. If the file is an Infile, Image is set to Blanks(Image.Length).

(2) If the new character set causes a change from EBCDIC to BIT or KEY or conversely, the appropriate character set changing character is written to the device using Breakoutimage. [Obviously, for Infiles a character is written only if the device is the terminal.]

(3) The current character set number is placed on the file character set stack and the character set number is set to the new character set.

When the procedure reset is executed, the following occurs:

(1) If the file is an Outfile or Printfile, then if Image.Strip NE NOTEXT, Outimage is called. If the file is an Infile, then Image is set to Blanks(Image.Length).

(2) If changing the character set to the top character set on the stack will cause a change from EBCDIC to BIT or KEY or conversely, the appropriate character set change character is written to the device using Breakoutimage. [Again, for Infiles, only the terminal character set is changed.]

(3) The character set number at the top of the stack replaces the character set number associated with the file and the stack is popped.

Partial implementations of the set procedures appear in the declaration of class out_file; the terminal character set changes are included; not all details match.

The file classes satisfy the class structure of file objects as described in the Common Base Definition and Simula BEGIN. That is, a class File is declared:

```
CLASS File (f_spec); TEXT f_spec;
BEGIN
    ...
END of File;
```

All attributes that are shared by all file objects are declared in class File. The specific file classes are declared with the following headings:

```
File CLASS Infile;
File CLASS Outfile;
Outfile CLASS Printfile;
```

A Printfile class video_file is declared with an additional boolean attribute blackout. If blackout is true, a video_file is just like a Printfile. If blackout is false, all lines written to the file are also written to the terminal with carriage control; see Section 7. The heading of the declaration is:

```
Printfile CLASS video_file
```

It is usually the case that two file objects (ttyin and tyyout) are associated with the terminal and in some circumstances there may be additional file objects associated with the terminal. This state of affairs leads to potential confusion concerning the actual character set of the terminal at any specific moment.

Most APL terminals can change the printed character set under program control and the specifications for the file objects require changing the terminal font. However, when two or more file objects refer to the terminal they may have a different view of the terminal type font and this may produce peculiar looking output on the terminal or unexpected interpretations of the input lines.

It is the responsibility of the file system to keep track of the type font that is in effect for the terminal. If the terminal type font is ASCII and a file object reads from or writes to the terminal in character set KEY or BIT then it is the responsibility of the file system to change the terminal type font before the read or write is performed.

For devices other than the terminal, it is the responsibility of the program to control the type font of the device. [It is assumed that there will only be one file object associated with other devices but this is not a requirement.]

More generally, execution may not be terminated for any error. If an error occurs, the appropriate message is written to the terminal and the user is asked for a corrective response or to select termination of execution.

5. Class Infile

Except when the CMS file specification is TTY:, class Infile will deal with end-of-file as in the system defined version.

That is, Endfile becomes true and Image is set to /*. Note carefully the specifications for Endfile; they are not what one immediately expects.

If the file specification is TTY:, an empty line will be treated as an empty line and Image will be set to Blanks(Image.Length). If the first character of an input line is control-Z, this line will be treated as an end-of-file.

At each call to Inimage, a line is read from the input device and translated into the appropriate character set. In addition, tabs are replaced by blanks under the assumption that tabs are set every eight spaces. If the result of expanding tabs in an input record is longer than Image.Length, the remaining text is retained and used to satisfy the next call to Inimage.

When the procedure Open is called, if the length of the parameter is less than the LRECL of the file, then the user may elect to extend Image to the appropriate length by answering a terminal prompt or elect to terminate execution. If Image.Length is greater than LRECL, input lines are padded with the appropriate number of blanks.

Instances of class Infile must be able to read any fixed or variable length record file in any format. The running program need not know anything about the file!

APL character set (BIT or KEY) is subject to three typographical conventions:

(1) When entering a line of input, characters may be entered in any order with arbitrary numbers of backspaces and spaces. The input line is interpreted just as it appears on the terminal.

(2) After a line of input is read, it is canonicalized to remove redundant spaces and so that strikeovers are of the form <ch1><bs><ch2> and such that Rank(<ch1>) <= Rank(<ch2>).

(3) After a line of input from the terminal only is canonicalized, it is examined for the character <lf> (ASCII 10, EBCDIC 37). If this character is found, the line is processed as follows:

    (i) All characters from the <lf> to the end of the line (including <lf>) are deleted.

    (ii) The remaining characters in the line are sent to the terminal using Breakout-image.

(iii) An additional line of input is read from the terminal.

(iv) A new string that consists of the remainder of the first line and the second input line are concatenated and processing continues at step (3)

All Infiles perform input canonicalization as described above for file character sets KEY and BIT. The appropriate transformation for Infiles with file character set APL is also performed.

6. Class Outfile

All files written by class Outfile are variable le??th record files with the shortest possible record length.

Each instance of Outfile is to have an internal variable which is true if the output is to be written with tabs replacing strings of blanks and false otherwise. This variable can be modified by means of the procedures tabs_on and tabs_off. When a call to one of these procedures is executed, the value of this variable is changed. By default, tabs are inserted into disk files and PUN: files and tabs are not inserted into TTY: and LPT: files; the appropriate initialization is performed when the class instance is created.

The procedure attribute Outtext is to be extended so that long text objects can be written over several lines as in class out_file in file OUTFILE SIMULA.

The procedure attribute Outimage is to be extended so that character set translation is performed in accord with the program and file character set. The text object Image.Strip after translation and tab processing is to be written to the output device.

The procedure Breakoutimage is an extension of Outfile. Character set translation in accord with the file and program character sets is performed. If the output device is TTY:, then the text Image.Sub(1,Image.Pos) is written to the terminal without a trailing <cr><lf> [carriage return -- line feed] and Image is set to Blanks(Image.Length). If the output device is not TTY:, then the text Image.Sub(1,Image.Pos) is to be added to an internal buffer. Successive calls to Breakoutimage for such files simply extend the buffer. When Outimage is next called, the buffer followed by the text to be written in response to the call to Outimage is written to the device.

Canonicalization of output in character sets KEY and BIT is not provided but in character set APL the output files are to be

suitable for printing with the APL print train.

7. Class Printfile

Class Printfile has all of the attributes of an Outfile because it is declared

```
Outfile CLASS Printfile;
```

Therefore, all of the statements about Outfiles apply to Printfiles.

Unless the output device is TTY:, the files written by Printfile are to be in a format suitable for printing with Fortran carriage control. Output carriage control may be written by the user program.

Except for output device TTY:, Printfiles control pagination by appending the character 1 to the beginning of lines that are to be printed on the top of a page. For output device TTY:, the module CLEAR is executed before writing a line that is to begin at the top of the next page.

8. Class Stream

Infile class stream is an extension of class Infile with two extensions of the procedure Inimage.

When Inimage is executed the first character of the input line is examined and if the character is an at sign (@), the remainder of the line is interpreted as a file specification including character set specification. If the character set specification is omitted, the character set specification of the current file is used.

This file specification is used to create another stream and input is taken from this stream until an end-of-file is encountered. After the end-of-file, the next input line from the current stream is processed.

Each call to Inimage does cause a line of input to be transmitted to Image.

If the first character of a line of input is not an at sign (@), then the value of Image is computed as follows.

(1) The first line is read from the file and trailing blanks are removed. If the result is a non-empty string, this string is placed in Image. If the result is an empty string, Image is set to Blanks(buf.Length) where the open call was Open(buf); steps 2 and 3 below are skipped.

(2) If the last character of Image.Strip is the minus sign (-) in the appropriate character set, then this character and the remaining blanks in Image are deleted. Another line of input is read from the file and added to the end of Image.

(3) Step 2 is repeated until a line whose last non-blank character is different from the minus sign is read. This line (with trailing blanks removed) is appended to Image.

This definition of the action of Inimage implies that Image.Length is different for different calls to Inimage.

In order to satisfy operating system restrictions, it will be necessary to impose some limit on the number of files that can be open at any one time. This limit is to be set as high as possible.

9. General Specifications

All advisory messages written to the terminal or file are to be enclosed in square brackets. Advisory message provide information but do not require a response. Messages that require a response are not enclosed in square brackets.

The first character of a message text will be either percent (%) or question mark (?). The character percent indicates that there is a possibility of an error and the character question mark indicates that a fatal error has occurred and execution must be terminated if it is not corrected. If execution is terminated because of an uncorrected error, the message

[? Fatal error, execution terminated.]

is printed on the terminal.

Terminal prompts which include messages are to use the query procedures in class dialog. If there is no message, the program is to prompt using Breakoutimage as follows: Primary input prompts use the character asterisk (*), secondary input prompts use the character sharp (#) and third level input prompts use the character at sign (@). CMS at monitor level prompts with period (.).

There are to be two predeclared and opened file objects: ttyin and ttyout which are the terminal as input (an Infile) and the terminal as output (an Outfile). All terminal dialog is to use these devices so that character set translation for messages can be performed. A program writing an error message should change the program character set to the appropriate set and then restore the character set to its previous value.

All identifiers declared in the file classes that are not attributes of the class will also be declared PROTECTED so that they are not available from outside the program. Similarly, variables whose value is of interest outside the class are to be declared PROTECTED and access to the variable is to be provided read-only by means of a simple procedure. This reduces the risk of a program changing something in the file classes in an unfortunate way.

Most of the file class code is to be written in SIMULA. Assembly procedures are to be used only if one of the following conditions are satisfied:

(1) It is not possible to write the code in SIMULA, e.g., the code to write to a file.

(2) An assembly coded procedure will be substantially more efficient than the corresponding SIMULA code. In this case a SIMULA coded version will also be provided.

The motivation for this specification is to make the program easier to maintain.

Program text is to be formatted with respect to the case of identifiers in accord with the default options of SIMED.

Each procedure or class declaration will have an end comment of the form:

```
END of <name>;
```

where <name> is the name of the procedure or class as formatted by SIMED. The information contained in the comment is, of course, helpful when reading the program. Following this format exactly makes it very much easier to edit program files.

All SIMULA programs will follow a consistent indentation convention. The indentation provided by SIMED or the indentation used in file OUTFILE SIMULA meets this requirement but other conventions are acceptable.

Comments are to be formatted as paragraphs with the first column of comment text beginning in column 8 (first ANSI tab) and the keyword comment beginning in column 1. The semi-colon terminating the comment will appear on a line alone in column 8. See the comment in procedure convert_to_apl in class out_file for an example.

Brief explanatory remarks may appear with the comment symbol exclamation point (!) in the program text. See procedures set_key_paired and set_ascii in class out_file for examples.

APPENDIX A

ASCII-EBCDIC TRANSLATION TABLES

USED AT

VIRGINIA TECH

The following pages are an extract from a Virginia Tech Center document that describes the ASCII-EBCDIC translation tables that are used with the Memorex 1380 Communications Processor on the VM/CMS System.

In order to get started on defining a new output translation table, I have attached a copy of a proposed table which would solve the problems with SCRIPT. This particular table may be only a starting point for deciding on a new table; however, whatever table we define must translate at least one EBCDIC character into each ASCII character.

The notations which I have used in the table are explained below.

The table contains 256 entries in a 16 by 16 array. Each entry contains two or three rows. The first row of an entry specifies an EBCDIC character; the second row specifies the ASCII character into which I propose that it be translated; and the third row specifies the ASCII character into which it is currently translated, which I obtained from a copy of the current table given to me by Mike Elliott. A second row containing only #'s indicates that I'm indifferent to how that character is translated (though that doesn't necessarily mean that a change is not in order). The third row is omitted if it is the same as the second, i.e., if I see no need to change the translation of that character. Each line of each entry, except as noted, consists of a hexadecimal code and an equals sign, followed by the character which the code represents (if it's a printing character) or the name of the character (if it's a control character). For the ASCII hexadecimal codes on the second lines of entries, I set the parity bit to 0; for the ASCII hexadecimal codes on the third lines of entries, I specified the parity bit (correct or not) as it is in the current table. Blanks after the equals sign indicate that that particular hexadecimal code has no assigned meaning. "sp" is used to indicate a space character, and "not" is used to indicate the EBCDIC "logical not". A vertical bar overstruck by an equals sign is used to indicate a character which cannot be printed on the terminal on which the table was printed, such as the "hook", "fork", and "chair" characters. A designation of "TN" at the end of a line indicates that that particular hexadecimal code prints as that character with the TN print train, but that that code does not otherwise represent any character.

My references for the character sets were System/370 Reference Summary (the "yellow card") and System/370 Principles of Operation, Appendix H: EBCDIC Chart.

The justification of the proposed translation of the EBCDIC cent sign (X'4A') to the ASCII caret (X'5E') should be mentioned: there is no ASCII cent sign, there is no EBCDIC caret, and each appears as shift/6 on its respective keyboard (at least on some).

There are a total of 18 changes in the proposed table.

Determining the impact of changing this translation is difficult. I would not anticipate any problems for those terminals or applications which use only printing characters and the basic control characters. The problems, if any, will most likely be with those

00=NUL 00=NUL 7F=DEL	01=SOH 01=SOH BA=:	02=STX 02=STX BA=:	03=ETX 03=ETX	04=PF 94=DC4	05=HT 09=HT 87=BEL	06=LC BA=:	07=DEL 7F=DEL	08=GE BA=:	09=RLF BA=:	0A=SMM BA=:	0B=VT 0B=VT 80=BS	0C=FF 0C=FF	0D=CR 0D=CR 84=EOT	0E=SO 0E=SO	0F=SI 0F=SI
10=DLE 10=DLE	11=DC1 11=DC1	12=DC2 12=DC2	13=IM 13=DC3	14=RES 80=NUL	15=NL 0C=FF	16=BS 08=BS	17=IL 80=NUL	18=CAN 18=CAN	19=EM 19=EM	1A=CC BA=:	1B=CU1 BA=:	1C=IFS 1C=FS	1D=IGS 1D=GS	1E=IRS 9E=RS	1F=IUS 1F=US
20=DS BA=:	21=SOS BA=:	22=FS 1C=FS 1A=SUB	23= BA=:	24=BYP 80=NUL	25=LF 0A=LF	26=ETB 17=ETB	27=ESC 1B=ESC	28= BA=:	29= BA=:	2A=SM BA=:	2B=CU2 BA=:	2C= BA=:	2D=ENQ 05=ENQ	2E=ACK 06=ACK	2F=BEL 07=BEL
30= BA=:	31= BA=:	32=SYN 16=SYN	33= BA=:	34=PN 92=DC2	35=RS 1E=RS 1A=SUB	36=UC BA=:	37=EOT 04=EOT 81=SOH	38= 8B=VT	39= BA=:	3A= BA=:	3B=CU3 BA=:	3C=DC4 14=DC4 13=DC3	3D=NAK 15=NAK BA=:	3E= BA=:	3F=SUB 1A=SUB
40=sp 20=sp	41= BA=:	42= BA=:	43= BA=:	44= BA=:	45= BA=:	46= BA=:	47= BA=:	48= BA=:	49= 5D=]	4A=cent 5E=^ 1A=SUB	4B=. 2E=.	4C=< 3C=<	4D=(26=(	4E=+ 2B=+	4F=| 7C=|
50=& 26=&	51= BA=:	52= BA=:	53= BA=:	54= BA=:	55= BA=:	56= BA=:	57= BA=:	58= BA=:	59= BA=:	5A=! 21=!	5B=$ 24=$	5C=* 2A=*	5D=) 29=)	5E=; 3B=;	5F=not 7E=~
60=- 2D=-	61=/ 2F=/	62= BA=:	63= BA=:	64= BA=:	65= BA=:	66= BA=:	67= BA=:	68= BA=:	69= BA=:	6A=¦ BA=:	6B=, 2C=,	6C=% 25=%	6D=_ 5F=_	6E=> 3E=>	6F=? 3F=?
70= BA=:	71= BA=:	72= BA=:	73= BA=:	74= BA=:	75= BA=:	76= BA=:	77= BA=:	78= BA=:	79=` 60=` 5B=[	7A=: 3A=:	7B=# 23=#	7C=@ 40=@	7D=' 27='	7E== 3D==	7F=" 22="
80= BA=:	81=a 61=a	82=b 62=b	83=c 63=c	84=d 64=d	85=e 65=e	86=f 66=f	87=g 67=g	88=h 68=h	89=i 69=i	8A= BA=:	8B=[TN 7B=[	8C= BA=:	8D= BA=:	8E= BA=:	8F= BA=:
90= BA=:	91=j 6A=j	92=k 6B=k	93=l 6C=l	94=m 6D=m	95=n 6E=n	96=o 6F=o	97=p 70=p	98=q 71=q	99=r 72=r	9A= BA=:	9B=]TN 7D=]	9C= BA=:	9D= BA=:	9E= BA=:	9F= BA=:
A0= BA=:	A1=~ BA=:	A2=s 73=s	A3=t 74=t	A4=u 75=u	A5=v 76=v	A6=w 77=w	A7=x 78=x	A8=y 79=y	A9=z 7A=z	AA= BA=:	AB= BA=:	AC= BA=:	AD=|TN 5B=| BA=:	AE= BA=:	AF= BA=:
B0= 5E=^	B1= BA=:	B2= BA=:	B3= BA=:	B4= BA=:	B5= BA=:	B6= BA=:	B7= BA=:	B8= BA=:	B9= BA=:	BA= BA=:	BB= BA=:	BC= BA=:	BD=]TN 5D=] 1A=SUB	BE= BA=:	BF= E0=
C0=| 7B=[BA=:	C1=A 41=A	C2=B 42=B	C3=C 43=C	C4=D 44=D	C5=E 45=E	C6=F 46=F	C7=G 47=G	C8=H 48=H	C9=I 49=I	CA= BA=:	CB= BA=:	CC=¦ BA=:	CD= BA=:	CE=¦ BA=:	CF= DC=\
D0=] 7D=] BA=:	D1=J 4A=J	D2=K 4B=K	D3=L 4C=L	D4=M 4D=M	D5=N 4E=N	D6=O 4F=O	D7=P 50=P	D8=Q 51=Q	D9=R 52=R	DA= BA=:	DB= BA=:	DC= BA=:	DD= BA=:	DE= BA=:	DF= E0=NUL
E0=\ 5C=\ BA=:	E1= BA=:	E2=S 53=S	E3=T 54=T	E4=U 55=U	E5=V 56=V	E6=W 57=W	E7=X 58=X	E8=Y 59=Y	E9=Z 5A=Z	EA= BA=:	EB= BA=:	EC=¦ BA=:	ED= BA=:	EE= BA=:	EF= BA=:
F0=0 30=0	F1=1 31=1	F2=2 32=2	F3=3 33=3	F4=4 34=4	F5=5 35=5	F6=6 36=6	F7=7 37=7	F8=8 38=8	F9=9 39=9	FA=¦ BA=:	FB= BA=:	FC= BA=:	FD= BA=:	FE= BA=:	FF=EO BA=:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	80	01	02	83	04	89	86	7F	08	89	8A	0B	8C	0D	0E	8F
1	10	91	92	13	94	15	08	97	98	19	1A	9B	1C	9D	9E	1F
2	20	A1	1C	23	A4	8A	97	9B	A8	29	2A	AB	2C	B5	86	07
3	B0	31	16	B3	34	9E	B6	01	38	B9	BA	3B	91	15	3E	1A
4	20	C1	C2	43	C4	45	46	C7	C8	5D	5E	AE	BC	A8	AB	7C
5	26	1	2	D3	54	D5	D6	57	58	D9	A1	A4	2A	29	3B	7E FE
6	AD	2F	62	E3	64	E5	E6	67	68	E9	7C	2C	25	DF	3E	BF
7	70	F1	F2	73	F4	75	76	F7	F8	E0	BA	23	40	A7	3D	A2
8	80	61	62	E3	64	E5	E6	67	68	E9	8A	FB	8C	0D	0E	8F
1	10	EA	6B	EC	6D	6E	EF	70	F1	F2	1A	FD	9C	9D	9E	9F
A	A0	FE	73	F4	75	76	F7	F8	79	7A	AA	AB	AC	5B	AE	AF
B	5E	31	B2	B3	34	B5	B6	B7	38	B9	BA	3B	BC	5D	2E	BF
C	FB	C1	C2	43	C4	45	46	07	C8	49	4A	CB	4C	CD	CE	4F
D	FD	4A	CB	4C	CD	CE	4F	D0	51	52	DA	5B	DC	5D	5E	DF
E	DC	61	D3	54	D5	D6	57	58	D9	DA	EA	6B	EC	6D	6E	EF
F	B0	21	29	B2	24	B5	B6	37	3A	B9	BA	FB	7C	FD	FE	7E

APPENDIX B

STAPL CONVENTION

FOR

ASCII/APL TERMINALS

(Reprinted from APL Quote Quad)

Standards

THE APL-ASCII OVERLAY STANDARD

(Editor's Note: The following description of APL-ASCII overlays for typewriter- and bit-paired terminals was originally written in 1972 by Lawrence M. Breed while he was at Scientific Time Sharing Corporation. The proposed standard was accepted around that time by the SHARE APL User's Group and is reproduced here by request of the STAPL Executive Committee with STSC's permission.)

Introduction

APL, a programming language developed by IBM and now offered by several manufacturers, is particularly well suited for interactive problem solving in both scientific and commercial areas. The use of APL is growing rapidly. More than 50 universities in North America are running APL systems; commercial APL service from a number of vendors has been doubling every nine months; and IBM has estimated that 70 percent of its own internal time sharing work is done with APL.

Until 1972, virtually all terminals used with APL systems were based on the IBM Selectric mechanism. Then, however, terminal manufacturers began to support APL features on ASCII terminals. Because APL uses a large number of graphic characters that do not occur in ASCII, an alternate character set was needed. Work toward a standard for APL characters on ASCII devices was first undertaken by James L. Ryan at Burroughs Corporation, and has since been advanced by other members of the APL Users Group. This standard was accepted by the APL Project of the SHARE organization and by STAPL; it is being used by many terminal manufacturers, computer manufacturers, and APL time sharing services.

The APL Keyboard

APL-ASCII is designed to "overlay" the keyboard of a 94-graphic ASCII terminal with a different set of graphics, without requiring any modification of electronic or electromechanical parts except for the character-generating mechanism itself (for example, a dot matrix read-only memory or print head). Specifically, pressing a key on the terminal causes exactly the same code to be transmitted, whether ASCII or APL graphics are employed. The ASCII codes 0/0 through 1/15 have the same significance in ASCII and in APL; similarly, the control-shift key positions are unchanged.

The de facto standard APL terminal is the IBM 2741 using print element number 1167987 (Correspondence coding) or 1167988 (BCD coding). Figure 1 shows the IBM 2741 APL keyboard. This is a particularly well laid out keyboard for APL use. The keyboard groups arithmetic, logical, and relational symbols, and places the parentheses and brackets near other paired punctuation characters. As far as possible, this keyboard associates alphabetic symbols and function symbols of similar shape or name; for example, over the I, O, and P are "iota", "circle", and "power". The orderly groupings and mnemonic pairings of the symbols are a significant aid to the user in finding and remembering symbol positions.

On any terminal that follows either the ANSI typewriter-pairing or bit-pairing standard, APL-ASCII provides a keyboard arrangement as close as possible to the IBM 2741 APL keyboard. The typewriter-pairing standard is preferred for APL-ASCII terminals. Figures 2 through 5 show the APL character set as it appears on the two ANSI keyboards, and in the ASCII transmission code. Figure 6 displays the ASCII transmission codes.

One side effect of producing nearly identical APL keyboards overlaying the two

ANSI keyboards is that the particular ASCII character overlaid by an APL character may depend on which ANSI keyboard arrangement is used. This is of no concern to the APL user, but must be resolved by the APL time sharing system. The first character of an APL sign-on is always a right parenthesis (ASCII code 2/2 on a typewriter-pairing terminal; 2/10 on a bit-pairing terminal). Once the time sharing system recognizes one of these two codes, it knows which code to use for every other APL character.

Some terminals deviate slightly from ANSI keyboard standards. If a terminal does not adhere to the standards for positions of ASCII characters, the transmission codes for APL characters must not be changed. The discrepancy must be reflected in the APL key positions, not the transmission codes.

Additional APL Characters

ASCII contains codes for 94 different graphics, or six more graphics than are on an APL Selectric typesphere. Six additional graphics in the APL-ASCII standard are:

$	Dollar Sign	◇	Diamond
⊢	Left Tack	⊣	Right Tack
{	Left Brace	}	Right Brace

Except for the diamond symbol, these characters have significance in APL only as printable graphic characters.

Use of Backspace and Overstruck Characters

Unlike other programming languages, APL makes heavy use of overstruck characters formed by typing one character, backspacing, and typing a second character. Two examples of APL overstrikes are ⌽ which is formed from ○ and |, and ! which is formed from . and '.

Backspacing does not delete input characters. Input correction is accomplished by backspacing to position the cursor, then depressing LINEFEED. Depressing LINEFEED has the effect of deleting all characters above and to the right of the cursor. Backspacing, then forward spacing, does not alter the meaning of any previously entered characters. In APL, entering the sequence

A,space,C,backspace,backspace,B,return

has the same effect as entering the sequence

A,B,C,return

These APL input editing rules are used by nearly all APL systems, and terminals that cannot physically backspace or terminals that erase buffered input when backspace is pressed are at a distinct disadvantage in the APL terminal market.

Miscellaneous

Replacing ASCII graphics by APL graphics in the terminal's print mechanism is the one important modification to an ASCII terminal to permit its use with APL. Most APL terminal manufacturers offer APL-engraved keycaps as well, at least as an option; however, APL users have been known to work successfully with just a picture of an APL keyboard propped up in front of them. One low cost alternative to engraving a new set of keycaps is to affix APL decals to the fronts of the keys. With this approach, both ASCII and APL characters can be shown on the keyboard without cluttering the keytops.

Several APL-ASCII terminals contain character generators for both APL and ASCII, so the terminal can be used easily in either environment. Selecting the desired code is done either with a switch on the terminal, or by transmission of Shift Out (SO) to shift from ASCII to APL, and Shift In (SI) to shift from APL to ASCII.

Summary

When the APL-ASCII standard was first proposed in 1972, it came at a very appropriate time. The variety of terminals that can support APL has increased

dramatically, and nearly all APL systems now support these terminals. Thus, agreement on APL codes and keyboards is important. Designers of all these terminals and systems have accepted APL-ASCII, because it calls for a minimum of re-education for the time sharing user and a minimum of change to terminal hardware.

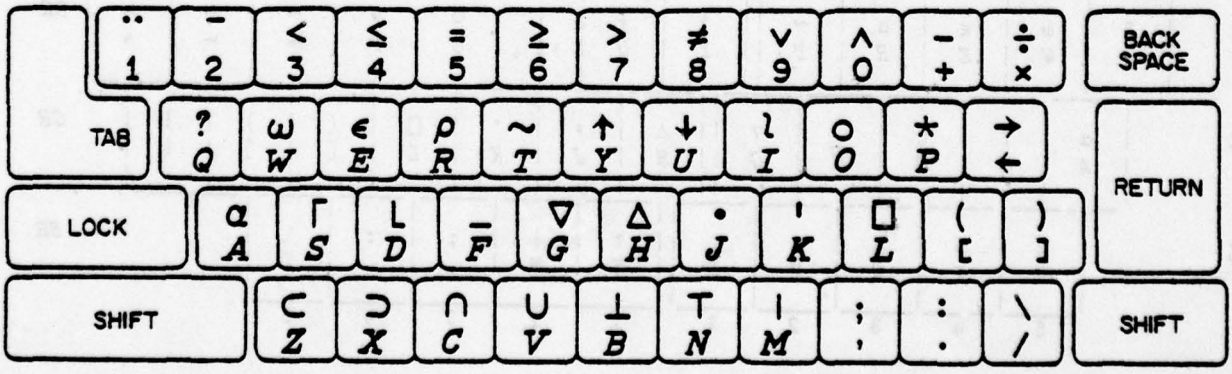


Figure 1. IBM 2741 APL Keyboard

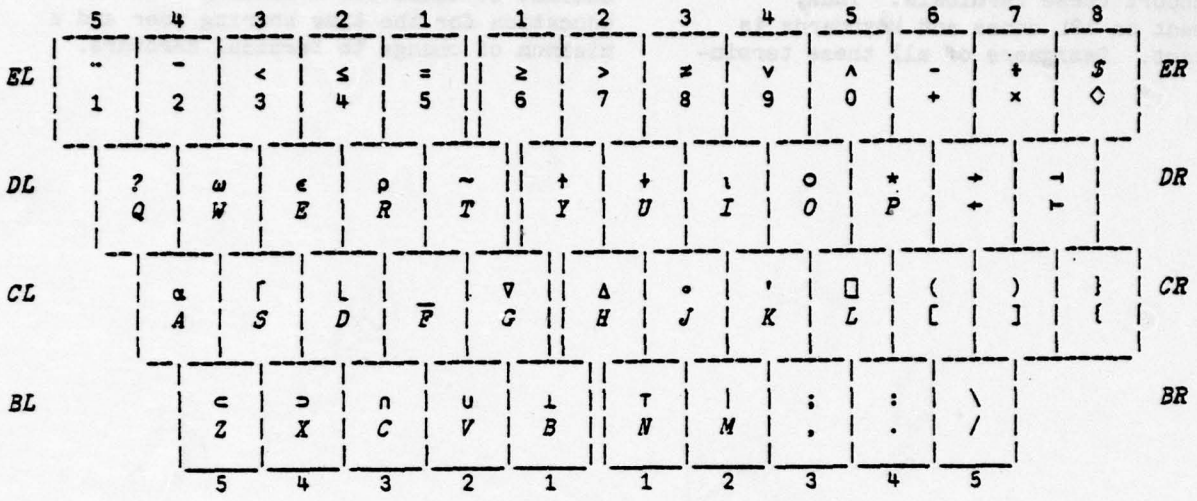


Figure 2. APL-ASCII Typewriter-Pairing Keyboard

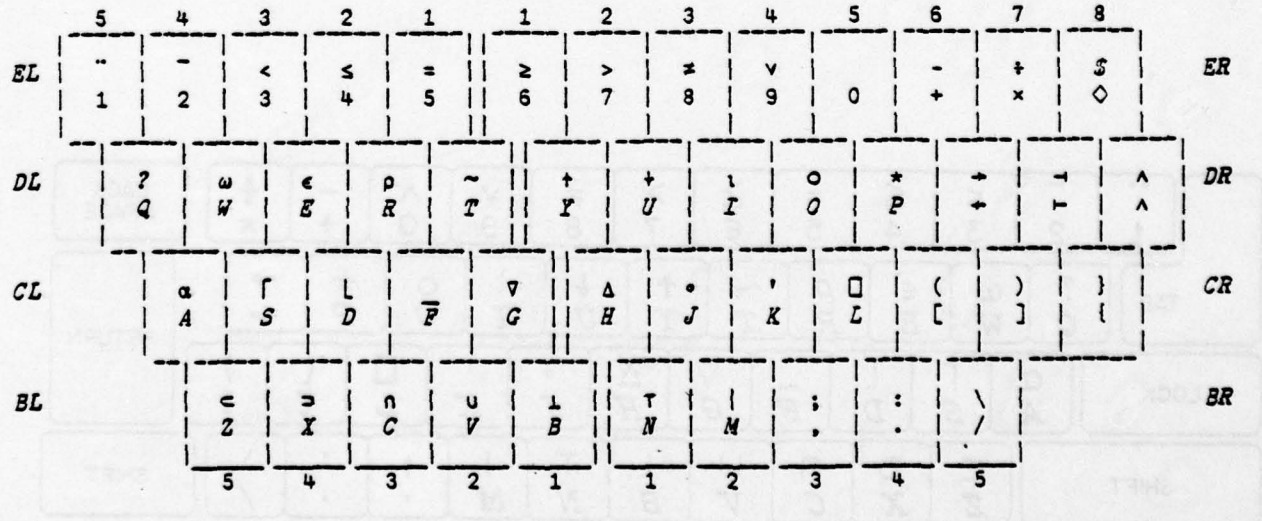


Figure 3. APL-ASCII Bit-Pairing Keyboard

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	¯	⋆	◇	P
1	SOH	DC1	¨	1	⍺	?	A	Q
2	STX	DC2	)	2	⊥	⍴	B	R
3	ETX	DC3	<	3	∩	⌈	C	S
4	EOT	DC4	≤	4	⌊	~	D	T
5	ENQ	NAK	=	5	∈	+	E	U
6	ACK	SYN	>	6	_	∪	F	V
7	BEL	ETB	]	7	∇	⍵	G	W
8	BS	CAN	∨	8	Δ	⊃	H	X
9	HT	EM	∧	9	⍳	↑	I	Y
10	LF	SUB	≠	(	∘	⊂	J	Z
11	VT	ESC	÷	[	'	←	K	{
12	FF	FS	,	;	⍞	⊢	L	⊣
13	CR	GS	+	×	|	→	M	}
14	SO	RS	.	:	⊤	≥	N	$
15	SI	US	/	\	○	-	O	DEL

Figure 4. APL-ASCII Typewriter-Pairing Transmission Codes

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	←	*	→	P
1	SOH	DC1	"	1	α	?	A	Q
2	STX	DC2	¯	2	⊥	ρ	B	R
3	ETX	DC3	<	3	∩	⌈	C	S
4	EOT	DC4	≤	4	⌊	~	D	T
5	ENQ	NAK	=	5	∈	+	E	U
6	ACK	SYN	≥	6	_	∪	F	V
7	BEL	ETB	>	7	∇	ω	G	W
8	BS	CAN	≠	8	Δ	⊃	H	X
9	HT	EM	∨	9	ι	↑	I	Y
10	LF	SUB	)	]	∘	⊂	J	Z
11	VT	ESC	(	[	'	⊢	K	⊣
12	FF	FS	,	;	□	◇	L	$
13	CR	GS	+	-	|	{	M	}
14	SO	RS	.	:	⊤	×	N	↓
15	SI	US	/	\	○	∧	O	DEL

Figure 5. APL-ASCII Bit-Pairing Transmission Codes

	0	1	2	3	4	5	6	7
0	*NUL*	*DLE*	*SP*	0	@	P	'	p
1	*SOH*	*DC1*	!	1	A	Q	a	q
2	*STX*	*DC2*	"	2	B	R	b	r
3	*ETX*	*DC3*	#	3	C	S	c	s
4	*EOT*	*DC4*	$	4	D	T	d	t
5	*ENQ*	*NAK*	%	5	E	U	e	u
6	*ACK*	*SYN*	&	6	F	V	f	v
7	*BEL*	*ETB*	'	7	G	W	g	w
8	*BS*	*CAN*	(	8	H	X	h	x
9	*HT*	*EM*	)	9	I	Y	i	y
10	*LF*	*SUB*	*	:	J	Z	j	z
11	*VT*	*ESC*	+	;	K	[	k	{
12	*FF*	*FS*	,	<	L	\	l	|
13	*CR*	*GS*	-	=	M	]	m	}
14	*SO*	*RS*	.	>	N	^	n	~
15	*SI*	*US*	/	?	O	_	o	*DEL*

Figure 6. ASCII Transmission Codes

APPENDIX C

ASCII to APL Translation

as used in

APLSF (Digital)

APL.MS (Spectra)

(Reprinted from the APL.MS Manual)

SECTION 9: SYSTEM COMMANDS

APL	TTY	ALTERNATE TTY	APL	TTY	APL	TTY
⍺	.AL	a	A	.ZA	⍀	.CB
⊥	.DE	b	B	.ZB	⍜	.CR
∩	.DU	c	C	.ZC	⌿	.CS
⌊	.FL	d	D	.ZC	⍇	.DQ
∈	.EP	e	E	.ZE	⍒	.GD
	.US	f	F	.ZF	⍋	.GU
⍛	.DL	g	G	.ZG	I	.IB
⍙	.LD	h	H	.ZH	●	.LG
⍳	.IO	i	I	.ZI	⍟	.NN
⍥	.SO	j	J	.ZJ	⍮	.NR
⍘	´	k	K	.ZK	⍗	.OU
⍞	.BX	l	L	.ZL	⍎	.PD
⍤	.AB	m	M	.ZM	⍈	.QQ
⍑	.EN	n	N	.ZN	⍊	.RV
⍜	.LO	o	O	.ZO	⍕	.TR
⍣	*	p	P	.ZP	=	=
⍰	?	q	Q	.ZQ	>	>
⍴	.RO	r	R	.ZR	<	<
⌈	.CE	s	S	.ZS	+	+
~	.NT	t	T	.ZT	-	-
⍦	.DA	u	U	.ZU	×	#
∪	.UU	v	V	.ZV	÷	%
⍵	.OM	w	W	.ZW	(	(
⊃	.LU	x	X	.ZX	)	)
⍎	^	y	Y	.ZY	[	[
⊂	.RU	z	Z	.ZZ	]	]
			Δ	.UD	/	/
			∇	.FO	\	\
			⍫	.EX	$	$
			←	.LA	,	,
			→	.GO	.	.
			≥	.GE	;	;
			≤	.LE	:	:
			≠	.NE	!	!
				.NG	^	&
			∨	.OR	^	"
				.DD		

not to be implemented

Figure 18, Teletype Mnemonics for A P L . M S

APPENDIX D

SOURCE LISTING OF

CLASS DIALOG

```
    COMMENT Copyright 1979 by Richard J. Orgass and Robert E. Porter.
            This program may be copied provided that this comment
            remains in the program text.

            Research sponsored by the Air Force Office of Scientific
            Research, Air Force Systems Command, under Grant No.
            AFOSR-79-0021.  The United States Government is authorized
            to reproduce and distribute reprints for Governmental
            purposes notwithstanding any copyright notation hereon.

            The information in this document is subject to change
            without notice.  The author, Virginia Polytechnic
            Institute and State University, the Commonwealth of
            Virginia and the United States Government assume no
            responsibility for errors that may be present in this
            program or its documentation.
            ;

    COMMENT A detailed description of this program is given in:

            DIALOG: A SIMULA class for writing interactive programs.
            Technical Memorandum No. 79-3, Department of Computer
            Science, Graduate Program in Northern Virginia, May 5, 1979.

            Copies may be obtained by writing to the first author at:

                    Richard J. Orgass
                    Department of Computer Science
                    VPI & SU
                    P. O. Box 17186
                    Washington, D. C. 20041
            ;
    CLASS dialog;

    BEGIN
```

```
TEXT PROCEDURE insert_tabs (t); VALUE t; TEXT t;

COMMENT The return value of this procedure is a text object
        that is the Strip of the tabulated version of its parameter.
        That is, spaces in the parameter text object are
        replaced with the HT character so as to minimize
        the space the text uses in storing displayable
        information.
        ;

    BEGIN
        TEXT out_t;
        INTEGER spaces;
        CHARACTER tab, curchar;

        t :- t.Strip;
        IF t = Blanks (t.Length)
            THEN BEGIN
                insert_tabs :- Blanks (1);
                GO TO exit;
                END;

        out_t :- Blanks (132);

COMMENT The character HT (Horizontal Tab) is ASCII 9, EBCDIC 5;

        tab := Char (5);
        spaces := 0;

        WHILE t.More DO
            BEGIN
                curchar := t.Getchar;
                IF Mod (t.Pos - 1, 8) = 0 AND spaces > 0
                    THEN BEGIN
                        out_t.Putchar (tab);
                        spaces := 0;
                        END;
                IF curchar = ' '
                    THEN spaces := spaces + 1
                    ELSE BEGIN
                        IF spaces > 0
                            THEN BEGIN
                                IF out_t.Pos + spaces >
```

```
                                        out_t.Length
                                         THEN GO TO done;
                                    out_t.SetPos
                                       (out_t.Pos + spaces);
                                    spaces := 0;
                                    END;
                               out_t.Putchar (curchar);
                               END;
                          END;

        done:     insert_tabs :- out_t.Strip;

        exit:     END of insert_tabs;
```

```
TEXT PROCEDURE expand_tabs (t); VALUE t; TEXT t;

COMMENT This procedure returns a text object which is the
        untabulated version of its text parameter.  That is,
        tab characters (HT) are replaced with sufficient spaces
        to make the text appear the same untabulated as it would
        look if the tabulated version were displayed on a terminal
        with "hard" tabs stops set every eisht positions.
        ;

    BEGIN

        TEXT out_t;
        INTEGER spaces;
        CHARACTER tab, curchar;

        t :- t.Strip;

COMMENT The HT (Horizontal Tab) character is ASCII 9, EBCDIC 5;

        tab := Char (5);
        out_t :- Blanks (132);

        WHILE t.More AND out_t.Pos < out_t.Length
            DO BEGIN
                curchar := t.Getchar;
                IF curchar = tab
                    THEN out_t.Setpos (8 * ((out_t.Pos + 8) // 8))
                    ELSE out_t.Putchar (curchar)
                END;

        expand_tabs :- out_t.Strip;

    END of expand_tabs;
```

```
COMMENT The return value of this procedure is a text object
        that is the concatenation of its two (text) parameters
        in the order first argument, second argument.;

TEXT PROCEDURE conc2(t1,t2); VALUE t1, t2; TEXT t1, t2;

BEGIN
    TEXT c;
    c:- Blanks(t1.Length+t2.Length);
    c.Sub(1,t1.Length) := t1;
    c.Sub(1+t1.Length,t2.Length) := t2;
    conc2 :- c
END of conc2;
```

```
BOOLEAN PROCEDURE no_blanks(t); TEXT t;
BEGIN
    INTEGER i, J;
    BOOLEAN answer;
    J := t.Length;
    answer := TRUE;
    FOR i := 1 STEP 1 UNTIL J DO
        answer := answer AND (t.Sub(i,1) NE " ");
    no_blanks := answer
END of no blanks;
```

```
TEXT PROCEDURE upcase(t); VALUE t; TEXT t;

COMMENT This procedure accepts a text object as a parameter
        and returns a new text object that is the same as its
        parameter except that lower case letters are changed
        into upper case letters.;

BEGIN
    INTEGER     i,
                char_code,
                len;

    CHARACTER c;

    TEXT new_string;

    len := t.Length;
    new_string :- Blanks(len);
    new_string.Setpos(1);

    FOR i := 1 STEP 1 UNTIL len DO
    BEGIN
        c := t.Getchar;
        char_code := Rank(c);
        IF (129 <= char_code AND char_code <= 137)
           OR (145 <= char_code AND char_code <= 153)
           OR (162 <= char_code AND char_code <= 169)

            THEN new_string.Putchar(Char(char_code+64))
            ELSE new_string.Putchar(c)
    END;

    upcase :- new_string

END of upcase;
```

```
TEXT PROCEDURE frontstrip(t); TEXT t;

COMMENT RETURNS A SUBTEXT OF T WITH LEADING BLANKS REMOVED;

BEGIN
    t.Setpos(1);
    WHILE t.More DO
    IF t.Getchar NE ' '
        THEN BEGIN
                frontstrip :- t.Sub(t.Pos-1,t.Length-t.Pos+2);
                t.Setpos(0)
             END;
END of frontstrip;



TEXT PROCEDURE rest(t); TEXT t;

COMMENT This procedure returns the portion of the text t
        that begins at t.pos and ends at t.length.  It
        is taken from the DEC-10 SIMULA manual.;

    IF t =/= NOTEXT
        THEN rest :- t.Sub(t.Pos,t.Length-t.Pos+1);



COMMENT This procedure accepts a parameter of type infile and
        closes the file and then reopens it with the same size
        image as it had before.  It is useful when an acceptable
        terminal response is a null line so that the end of
        file on the input stream does not cause an error termination.;

PROCEDURE next_file(device); REF(Infile) device;

BEGIN
    INTEGER i;
    i := device.Image.Length;
    device.Close;
    device.Open(Blanks(i))
END of next_file;
```

```
BOOLEAN PROCEDURE is_integer(t); NAME t; TEXT t;

COMMENT This procedure returns TRUE if its text parameter
        contains an integer and FALSE otherwise.;

BEGIN

    INTEGER     i,
                len;

    BOOLEAN     answer;

    TEXT        temp_t;

    CHARACTER   c;

    temp_t :- Blanks(200);
    temp_t := t;
    temp_t :- frontstrip(temp_t.Strip);
    temp_t.Setpos(1);
    len := temp_t.Length;
    c := temp_t.Getchar;
    answer := c = '+' OR c = '-' OR Digit(c);

    FOR i := 2 STEP 1 UNTIL len DO
    BEGIN
        c := temp_t.Getchar;
        answer := answer AND Digit(c)
    END;

    is_integer := answer

END of is_integer;
```

```
PROCEDURE initialize_terminal;
BEGIN
    cpcommand("TERM PROMPT OFF");
    cpcommand("TERM LINEND OFF");
    cpcommand("TERM LINES 255");
    cpcommand("SET WNG OFF");
    cpcommand("SET ACNT OFF");
    cpcommand("SET MSG OFF");
    Sysin.Close;
    cmscommand("FILEDEF SYSIN CLEAR");
    cmscommand("FILEDEF SYSIN TERM (LOWCASE");
    Sysin.Open(Blanks(80));
    tty :- NEW out_file("TTY:");
    tty.Open(Blanks(255));
END of initialize_terminal;



PROCEDURE restore_terminal;
BEGIN
    TEXT        command;

    cpcommand("TERM PROMPT .");
    command :- Copy("TERM LINEND  ");
    command.Sub(command.Length,1).Putchar(Char(39));
    cpcommand(command);
    cpcommand("SET WNG ON");
    cpcommand("SET MSG ON");
END of restore_terminal;



PROCEDURE freeze(returncode); NAME returncode; INTEGER returncode;
INSPECT tty DO
BEGIN
    Outtext("freeze is not available."); Outimage;
    Outtext("execution continued."); Outimage
END of freeze;
```

```
REF(Outfile) PROCEDURE find_outfile(t); NAME t; TEXT t;

COMMENT This procedure accepts a text parameter that consists
        of a <fn> <ft> optionally followed by a <ft>.  It
        issues a filedef using a unique DDNAME and then uses
        this DDNAME to create a new Outfile which is its return
        value.  The text parameter is converted to upper case.
        ;
BEGIN
    INTEGER lrecl ;

    lrecl := 120 ;
    find_outfile :- NEW Outfile(set_dd_output(t,lrecl));

END of find_outfile;
```

```
REF(Infile) PROCEDURE find_infile(t); NAME t; TEXT t;

COMMENT This procedure accepts a file specification consisting
        of at least <fn> as a parameter.  Using the CMS
        command STATE, it determines if the file exists.  If
        the file exists, a unique DD name is assigned to the
        file via a FILEDEF command.  This DD name is used to
        create a NEW Infile and this Infile is the return
        value of the procedure.

        If there is an error in the file name that is the
        parameter of find_infile, appropriate error messages
        are printed and a corrected file name is read from
        the terminal.
        ;

BEGIN
    REF(Infile) file ;
    INTEGER lrecl ;

    file :- NEW Infile(set_dd_input(t,lrecl)) ;
    find_infile :- file  ;

END of find_infile;
```

```
TEXT PROCEDURE text_request(prompt, default, help);
        NAME prompt, default, help;
        TEXT prompt, default;
        BOOLEAN help;

COMMENT This procedure issues the parameter prompt as a message
        to the terminal.  If default is a text object other than
        NOTEXT, this default is printed after the prompt enclosed
        by DEFAULTCHARs and followed by a PROMPTCHAR.  The user
        response is read from the terminal and the following
        actions are taken:

        (1) If the response consists of the character ?, then
            the help procedure is called to print a help message
            on the terminal.  Input is again requested.

        (2) If the response is an empty line (end-of-file in CMS),
            then the default is the return value provided that it
            is not NOTEXT.  If the default is NOTEXT, an error
            message is issued and the terminal is prompted again.

        (3) If the response is any other string, it becomes the
            return value of the procedure.


WARNING:  Inputs that consist of only a question mark (?) or
        only the characters "/*" cannot be accepted by this
        procedure.
        ;

INSPECT tty DO
BEGIN
    TEXT t;
    INTEGER case_no;
    BOOLEAN bad_input;
    TEXT temp_default;
    SWITCH case := help_output,blank_line,input;

    temp_default :- Blanks(200);
    temp_default := default;
    temp_default :- temp_default.Strip;
    bad_input := TRUE;
    GO TO request;
```

```
            WHILE bad_input DO
            BEGIN

                IF case_no NE 1
                  THEN BEGIN
                       Outtext("? Default value may not be selected. ");
                          Outtext("Please try again.");
                          Outimage
                       END;
                request:

                IF prompt = NOTEXT
                   THEN Outchar(term_prompt)
                   ELSE Outtext(prompt);
                IF temp_default =/= NOTEXT
                   THEN BEGIN
                            Outchar(default_char);
                            Outtext(temp_default);
                            Outchar(default_char);
                            Outchar(prompt_char);
                       END;
                Outchar(' ');
                Breakoutimage;

                Inimage;
                t :- Sysin.Image.Strip;
                          IF t.Length = 0
                             THEN case_no := 2
                             ELSE IF t.Length=1
                                     THEN BEGIN
                                             IF t.Sub(1,1)="?"
                                                THEN case_no := 1
                                                ELSE case_no := 3
                                          END
                                  ELSE IF t.Length=2 AND
                                          t.Sub(1,2)="/*"
                                             THEN case_no := 2
                                             ELSE case_no := 3;
                GO TO case(case_no);

                help_output:

                    IF help
```

```
                    THEN Outtext("No help available.");
                Outimage;
                GO TO exit_case;

            blank_line:

                next_file(Sysin);
                IF temp_default =/= NOTEXT
                    THEN BEGIN
                            t :- temp_default;
                            bad_input := FALSE
                         END;
                GO TO exit_case;

            input:

                bad_input := FALSE;

            exit_case:

        END bad_input loop;

        text_request :- Copy(t)

    END of text_request;
```

```
BOOLEAN PROCEDURE boolean_request(prompt,default,help);
        NAME prompt, help;
        VALUE default;
        TEXT prompt;
        BOOLEAN default, help;
COMMENT This procedure prompts the terminal with a question
        that requires a yes or no answer.  If the response
        is y or Y then the return value is TRUE.  If the
        response is n or N the return value is FALSE.
        The parameter prompt is the text string that is to be
        displayed on the terminal as the prompt.  The parameter
        default is TRUE or FALSE (keywords) and the parameter
        help is a procedure to print a help message.
        ;

INSPECT tty DO
BEGIN
    BOOLEAN     t,
                bad_input,
                no_help;

    INTEGER case_no;

    TEXT u;

    SWITCH case := help_output, blank_line, input;

    bad_input := no_help := TRUE;
    GO TO request;

    WHILE bad_input DO
    BEGIN
        IF bad_input AND no_help
            THEN BEGIN
                    Outtext("? Please answer y or n.");
                    Outimage
                 END;

    request:
        IF prompt = NOTEXT
            THEN Outchar(term_prompt)
            ELSE Outtext(prompt);
        Outchar(default_char);
```

```
                IF default
                    THEN Outchar('y')
                    ELSE Outchar('n');
                Outchar(default_char);
                Outchar(prompt_char);
                Outchar(' ');
                Breakoutimage;

                Inimage;
                u :- upcase(frontstrip(Sysin.Image.Strip));
                case_no := IF u.Length = 0
                              THEN 2
                              ELSE IF u = "?"
                                      THEN 1
                                      ELSE IF u = "/*"
                                              THEN 2
                                              ELSE 3;
                GO TO case(case_no);
```

```
            help_output:

                IF help
                    THEN Outtext("No help available.");
                Outimage;
                no_help := FALSE;
                bad_input := TRUE;
                GO TO exit_case;

            blank_line:

                next_file(Sysin);
                t := default;
                bad_input := FALSE;
                no_help := TRUE;
                GO TO exit_case;

            input:

                IF u.Sub(1,1) = "Y"
                    THEN BEGIN
                            t := TRUE;
                            bad_input := FALSE;
                            no_help := TRUE;
                         END
                    ELSE IF u.Sub(1,1) = "N"
                            THEN BEGIN
                                    t := FALSE;
                                    bad_input := FALSE;
                                    no_help := FALSE;
                                 END
                            ELSE BEGIN
                                    bad_input := TRUE;
                                    no_help := TRUE;
                                 END;

            exit_case:
        END;

        boolean_request := t

    END of boolean_request;
```

```
INTEGER PROCEDURE integer_request(prompt,default,min,max,help);
        NAME prompt, help;
        VALUE default, min, max;
        TEXT prompt;
        INTEGER default, max, min;
        BOOLEAN help;

COMMENT This procedure prints the first parameter on the terminal
        followed by the second enclosed in default_chars and then
        reads a line of input.  If the input is an integer between
        min and max, then this integer is the return value of the
        procedure.  If the input is not an integer or is not in
        the specified range, then an error message is issued
        and another input is requested.  If the response is an
        empty line, the return value is default and if a ? is
        the response, then the procedure help is executed.
        ;

INSPECT tty DO
BEGIN
    INTEGER     t,
                case_no;

    TEXT        u,
                int;

    BOOLEAN     no_help,
                bad_syntax,
                bad_answer;

    SWITCH case :=      help_request,
                        blank_line,
                        input;

    no_help := bad_syntax := bad_answer := TRUE;
    GO TO request;
```

```
        WHILE bad_syntax OR bad_answer OR (NOT no_help) DO
        BEGIN
            IF bad_syntax AND no_help
                THEN BEGIN
                        Outtext("? The input was not an integer. ");
                        Outtext("Please try again.");
                        Outimage
                    END;

            IF bad_answer AND no_help
                THEN BEGIN
                        Outtext("? The input integer was out of the ");
                        Outtext("acceptabe range: [");
                        int :- Blanks(5);
                        int.Putint(min);
                        int :- frontstrip(int);
                        Outtext(int);
                        Outchar(',');
                        int :- Blanks(5);
                        int.Putint(max);
                        int :- frontstrip(int);
                        Outtext(int);
                        Outtext("]."); Outimage;
                        Outtext("Please try again."); Outimage
                    END;

            request:

            IF prompt = NOTEXT
                THEN Outchar(term_prompt)
                ELSE Outtext(prompt);
            Outchar(default_char);
            int :- Blanks(5);
            int.Putint(default);
            int :- frontstrip(int);
            Outtext(int);
            Outchar(default_char);
            Outchar(prompt_char);
            Outchar(' ');
            Breakoutimage;

            Inimage;
            u :- frontstrip(Sysin.Image.Strip);
```

```
        u := u;  COMMENT PATCH TO GET AROUND COMPILER BUG.;
                 case_no := IF u.Length = 0
                              THEN 2
                              ELSE IF u = "?"
                                     THEN 1
                                     ELSE IF u = "/*"
                                            THEN 2
                                            ELSE 3;
                 GO TO case(case_no);
```

```
            help_request:

                IF help
                    THEN Outtext("No help available.");
                Outimage;
                no_help := FALSE;
                bad_syntax := TRUE;
                bad_answer := TRUE;
                GO TO exit_case;

            blank_line:

                next_file(Sysin);
                t := default;
                IF min <= t AND t <= max
                    THEN BEGIN
                            bad_answer := FALSE;
                            t := default
                         END
                    ELSE BEGIN
                            bad_answer := TRUE;
                            Outtext("Invalid default value. ");
                            Outtext("Please provide correction.");
                            Outimage
                         END;
                bad_syntax := FALSE;
                no_help := TRUE;
                GO TO exit_case;

            input:

                IF is_integer(u)
                    THEN BEGIN
                            bad_syntax := FALSE;
                            t := u.Getint;
                            bad_answer := NOT (min<=t AND t<=max)
                         END
                    ELSE BEGIN
                            bad_syntax := TRUE;
                            bad_answer := FALSE;
                         END;
                no_help := TRUE;
```

```
            exit_case: ;

        END;
        integer_request := t;

    END of integer_request;
```

```
REF(Infile) PROCEDURE set_infile(t); NAME t; TEXT t;

COMMENT The parameter of this procedure is to have as its value
        the <fn> [<ft> [<fm>]] of a file that is to be read. If the
        file exists, a unique DDNAME is assigned and a filedef for
        this file is issued. This filedef includes RECFM and LRECL.
        This unique DDNAME is used to create a new instance of
        class Infile. After opening this Infile with an argument
        Blanks(LRECL), the Infile is returned. Note that the file
        object has been opened.

        NOTE:

        The effect of this statement is that it is possible to open
        any CMS file and read it without any knowledge of the file
        description!

        If the file specification that is the parameter is in error,
        the user is informed of the error and asked for another file
        specification. At this point, the user may respond with
        CMS to enter CMS commands in the subset that does not
        destroy the core image. He may return to the program
        by typing the command return.
        ;

BEGIN
    INTEGER lrecl ;
   REF(Infile) file ;


   file :- NEW Infile(set_dd_input(t,lrecl)) ;
   file.Open(Blanks(lrecl)) ;
   set_infile :- file ;

END of set_infile;
```

```
REF(Outfile) PROCEDURE set_outfile(file_name,lrecl);

NAME file_name ;
TEXT file_name ;   INTEGER lrecl ;

COMMENT The parameter FILE_NAME is to have as its value the
        <fn> [ <ft> [ <fm> ]] of an output file.  A unique
        DD name is assigned to the file and a cms FILEDEF is
        issued for it.  The file will be a fixed record format
        file with logical record length of LRECL.

        The file is opened.                                    ;


BEGIN
    REF(Outfile) file ;

    file :- NEW Outfile(set_dd_output(file_name,lrecl)) ;
    file.Open(Blanks(lrecl)) ;
    set_outfile :- file ;

END of set_outfile ;
```

```
REF(Printfile) PROCEDURE set_printfile(file_name,lrecl) ;
NAME file_name ;
TEXT file_name ; INTEGER lrecl ;

COMMENT  For the file name in FILE_NAME, issue a CMS filedef
         with a fixed record format of logical record length
         of LRECL.  FILE_NAME must have the following format:
         <fn> [ <ft> [ <fm> ]].

         The file is opened and the class printfile ref is returned. ;


BEGIN
    REF(Printfile) file ;

    file :- NEW Printfile(set_dd_output(file_name,lrecl)) ;
    file.Open(Blanks(lrecl)) ;
    set_printfile :- file ;

END of set_printfile ;
```

```
PROCEDURE cms_subset;

COMMENT This procedure accepts input from the terminal and passes
        the input lines to CMS for execution.  The terminal prompt
        is set to # at entry and to * at exit.  Appropriate messages
        are printed on the terminal.
        ;

INSPECT tty DO
BEGIN
    INTEGER retcode;
    TEXT command;

    Outtext("[Entering CMS subset.]");
    Outimage;
    Outchar('#');
    Breakoutimage;

    tty_inimage(Sysin);
    command :- upcase(frontstrip(Sysin.Image.Strip));

    WHILE command NE "RETURN" DO
    BEGIN
        cmscommand(command,retcode);
        Outchar('R');
        IF retcode NE 0
            THEN BEGIN
                    Outchar('(');
                    Outint(retcode,3);
                    Outchar(')')
                END;
        Outchar(';');
        Outimage;
        Outchar('#');
        Breakoutimage;
        tty_inimage(Sysin);
        command :- upcase(frontstrip(Sysin.Image.Strip));

    END of command loop;

    Outtext("[Returning to Simula program.]");
    Outimage;
```

```
END of cms_subset;
```

```
EXTERNAL assembly PROCEDURE cmscommand,
                            cpcommand,
                            error,
                            Maxint;

CHARACTER   default_char,
            help_char,
            prompt_char,
            term_prompt;

INTEGER     dd_count;

REF(out_file) tty;


  COMMENT

  GLOBAL VARIABLES used by the filedef procedures:

  DDCOUNT - Used in selecting a ddname by SIMXXX procedure.
  Last integer part selected.

  MARK_CHAR - Character used to indicate illegal characters
  by the procedure PART.
  May be changed for different character sets.  ;

  INTEGER ddcount ;
  CHARACTER mark_char ;
```

```
        COMMENT

        If the file id supplied in FILE_NAME is free of syntax errors,
        issue a filedef for it using LENGTH for the lrecl and F for
        recfm.  Return the ddname selected by SIMXXX.  ;

        TEXT PROCEDURE set_dd_output(file_name,lrecl);
        VALUE file_name ;
        NAME lrecl ;
        TEXT file_name;
        INTEGER lrecl ;

      INSPECT tty DO
        BEGIN
          BOOLEAN ok,error ;
          INTEGER n,m,l,retcode ;
          TEXT  empty,title,cmd,ddname,recfm,temp,options  ;
          TEXT ARRAY token(1:10) ;

          title :- Copy("from set_dd_output: ");
          empty :- NOTEXT ;

      COMMENT   Break up file id into tokens and check for syntax. ;
          rats: ok := TRUE ;
          FOR n := 1 STEP 1 UNTIL 10 DO BEGIN
             token(n) :- part(title,error,file_name);
             IF  error THEN ok := FALSE ;
          END;


      COMMENT If the user response was CMS: then enter CMS_SUBSET  ;


        IF token(1) = "CMS:"
          THEN BEGIN
              cms_subset ;
              file_name :- text_request("file id? ","CMS:",fdhelp);
              GO TO rats ;
           END ;


      COMMENT If user response was "TTY:" then he wishes
              output to terminal.  ;
```

```
        IF token(1) ^= "TTY:"
           THEN BEGIN

        COMMENT  Set the default file type and mode. ;

                 IF token(2) = empty THEN token(2) :- Copy("LOG");
                 IF token(3) = empty THEN token(3) :- Copy("A");

        COMMENT  Check for missing file name part.  No default allowed. ;

                 IF token(1) = empty
                 THEN BEGIN
                    ok:=FALSE;
                    Outtext(title);
                    Outimage;
                    Outtext("? missing file name part of file id in '");
                    Outtext(file_name);
                    Outtext("'");
                    Outimage;
                 END;

        COMMENT  If there was an error, request a correction from
                 the user.  ;


                 IF NOT ok
                 THEN BEGIN
                    file_name:-text_request("file id? ","CMS:",fdhelp);
                    GO TO rats;
                 END ;

        COMMENT Build the file id from the first three tokens.  ;

                 temp :- Blanks(80) ;
                 m := 1 ;
                 FOR n := 1 STEP 1 UNTIL 3 DO BEGIN
                    l := token(n).Length ;
                    temp.Sub(m,l) := token(n) ;
                    m := m + l + 1 ;
```

```
            END ;
            recfm :- Copy("F");
            file_name :- Copy(temp.Strip) ;

        COMMENT  Build the options from the remaining tokens.  ;


            n:= 4;
            options:-Blanks(80);
            m := 1 ;
            WHILE token(n) ^= empty DO BEGIN
               l := token(n).Length ;
               options.Sub(m,l) := token(n) ;
               m := m + l + 1;
               n := n + 1
            END;
            options :- options.Strip;

        END
        ELSE  BEGIN
          file_name :- token(1) ;
          options :- Copy("LOWCASE") ;
          lrecl := 80 ;
        END of TTY case ;
        COMMENT  Issue a FILEDEF for the file with ddname = DDNAME.  ;

            ddname :- simxxx ;
            IF NOT filedef(file_name,ddname,recfm,lrecl,options)
            THEN BEGIN
            Outtext(title);
            Outimage ;
            file_name :- text_request("file id? ","CMS:",fdhelp);
            GO TO rats;
          END;

        COMMENT Success.  Filedef completed.  Return the ddname to the
                calling program.   ;


            set_dd_output :- ddname ;

          END of set_dd_output;
```

```
        COMMENT

           If the file id supplied in FILE_NAME is free of syntax errors
           and if the file is present then read its lrecl and recfm and
           issue a filedef for it with the ddname selected by SIMXXX.
           If there are any problems, print an error message and request
           a correction file id from the user.  Return the selected ddname
           and set LENGTH to the lrecl.  ;


           TEXT PROCEDURE set_dd_input(file_name,lrecl);
           VALUE file_name ;
           NAME lrecl ;
           TEXT file_name;
           INTEGER lrecl ;

        INSPECT tty DO
           BEGIN
              BOOLEAN ok,error ;
              INTEGER n,m,l,retcode ;
              TEXT empty,title,cmd,ddname,recfm,temp,options,ans  ;
              CHARACTER c;
              TEXT ARRAY token(1:10) ;

              title :- Copy("from set_dd_input: ");
              empty :- NOTEXT ;

        COMMENT  Check the file id for syntax.  ;

              rats: ok := TRUE ;
              FOR n := 1 STEP 1 UNTIL 10 DO BEGIN
                 token(n) :- part(title,error,file_name);
                 IF  error THEN ok := FALSE ;
              END;


        COMMENT  If the user responsed with "CMS:" then enter CMS_SUBSET    ;

          IF token(1) = "CMS:"
             THEN BEGIN
                    cms_subset ;
                    file_name :- text_request("file id? ","CMS:",fdhelp);
```

```
                GO TO rats ;
        END ;

    COMMENT If the user responsed with "TTY:" then he wishes a
            terminal instead of a disk file .                    ;

      IF token(1) ^= "TTY:"
         THEN BEGIN

    COMMENT  Set the default file type and modes.  ;

          IF token(2) = empty THEN token(2) :- Copy("DATA");
          IF token(3) = empty THEN token(3) :- Copy("A");

    COMMENT  Check for missing file name part.  No default allowed. ;

          IF token(1) = empty
          THEN BEGIN
             ok:=FALSE;
             Outtext(title);
             Outimage;
             Outtext("? missing file name part of file id in '");
             Outtext(file_name);
             Outtext("'");
             Outimage;
          END;

    COMMENT  Check to see if the file exists.  It must exist before return.
             STATE will return a 0 for the return code if the file
             exist.  ;

          ext:  IF ok THEN BEGIN
             cmd :- Blanks(80);
             cmd.Sub(1,10):="STATE" ;
             cmd.Setpos(11);
             m := 11 ;
             FOR n := 1 STEP 1 UNTIL 3 DO BEGIN
                1:= token(n).Length ;
                cmd.Sub(m,1) := token(n);
```

```
            m := m + 1 + 1 ;
         END ;


         cmscommand(cmd,retcode) ;
         IF retcode ^= 0
         THEN BEGIN
            Outtext(title);
            Outimage;
            Outtext("file '");
            Outtext(file_name);
            Outtext("' not found");
            Outimage;
            ok := FALSE ;
         END ;

      END;

   COMMENT  If there has been in errors above request a correction
            from the user.  ;


      IF NOT ok
      THEN BEGIN
         file_name:-text_request("file id?","CMS:",fdhelp);
         GO TO rats;
      END ;


   COMMENT  At this point the file id is correct and the file exists.
            Using the stack option of LISTFILE, the RECFM and
            LRECL of the file are stacked on the console stack. ;


      file_name:-cmd.Sub(11,70);
      file_name:-file_name.Strip;

      cmd.Sub(1,8) := "LISTFILE" ;
      cmd.Sub(file_name.Length+12,9) := "(FO STACK" ;
      cmscommand(cmd,retcode) ;

   COMMENT  If the return code is not zero, the file is on an
            extension of the "A" disk.  Nothing was stacked.
```

```
            Change the file mode to '*' and do it again.
            This time several lines may be stacked.  ;


         IF retcode ^= 0 THEN BEGIN
            token(3) :- Copy("*") ;
            file_name := NOTEXT ;
            file_name.Setpos(1) ;
            m := 1 ;
            FOR n := 1 STEP 1 UNTIL 3 DO BEGIN
               l := token(n).Length ;
               file_name.Sub(m,l) := token(n) ;
               m := m + l + 1 ;
            END ;
            cmscommand(cmd,retcode) ;
         END;

      COMMENT  Read the first record for the files RECFM and LRECL.
               Using DESBUF to clear any remaining lines from the
               console stack.  ;


         Inimage ;
      COMMENT       CMSCOMMAND("DESBUF",RETCODE) ;
         temp :- Intext(70) ;
         temp.Setpos(22);
         recfm :- part(title,error,temp);
         Sysin.Image.Setpos(26) ;
         lrecl := Inint ;

      COMMENT At this point the file id is correct and exist.  The
              files RECFM and LRECL have been obtained for the filedef. ;


      COMMENT  Build the options from the remaining tokens.  ;


         n:= 4;
         options:-Blanks(80);
         m := 1 ;
         WHILE token(n) ^= empty DO BEGIN
            l := token(n).Length ;
```

```
            options.Sub(m,1) := token(n) ;
            m := m + 1 + 1;
            n := n + 1
          END;
          options :- options.Strip;

       END
       ELSE  BEGIN
           file_name :- token(1) ;
           options :- Copy("LOWCASE") ;
          lrecl := 80 ;
       END of the TTY case ;


     COMMENT Issue a filedef.  If there are problems request a
             correction from the user and try again.  ;


          ddname :- simxxx ;
          IF NOT filedef(file_name,ddname,recfm,lrecl,options)
          THEN BEGIN
          file_name :- text_request(title,"CMS:",fdhelp);
          GO TO rats ;
        END ;


     COMMENT  Return to ddname: DDNAME, to the calling program and
              set the length via the parameter list.  ;


       set_dd_input :- ddname ;

     END of set_dd_input;
```

```
COMMENT

This procedure translates lowercase letters to uppercase.  ;


CHARACTER PROCEDURE xlate(c) ;
VALUE c ; CHARACTER c ;
BEGIN
   IF c >= 'a' AND c <= 'i'  OR
   c >= 'j' AND c <= 'r'  OR
   c >= 's' AND c <= 'z'
   THEN xlate := Char(Rank(c) + 64)
   ELSE xlate := c
END of xlate;
```

```
COMMENT

Each call upon PART produces a tokenlike object.  Valid characters
in the object are letters and digits only.  Lowercase letters are
mapped into uppercase.  Objects may be seperated by any number
of blanks or a single period.
If an illegal character is found, an error message is printed
and the offending character is indicated.  An error response is
returned to the calling program.  Use of '.' can generate empty
tokens which cause a NOTEXT object being returned.  Also additional
calls after the end of the orisional string has been reached
return the same: NOTEXT.

The global character variable MARK_CHAR is used to indicate
the location of the illegal character.  This may be changed
when using a different character set.  ;


TEXT PROCEDURE part(title,error_flag,xname) ;
VALUE title ;
NAME xname; TEXT xname,title ;
BOOLEAN error_flag ;
INSPECT tty DO
BEGIN
   CHARACTER c ;
   BOOLEAN error ;
   TEXT yname ;



   yname:-Blanks(xname.Length);
   c := ' ' ;
   WHILE xname.More AND c = ' ' DO c := xname.Getchar ;
   IF c =/= ' ' THEN xname.Setpos(xname.Pos - 1 );

   IF xname.More
   THEN BEGIN

      error_flag := FALSE ;
      error := FALSE ;
```

```
            WHILE NOT error AND xname.More  DO
            BEGIN c := xlate(xname.Getchar) ;
               IF NOT(Letter(c) OR Digit(c))
               THEN error := TRUE ;
            yname.Putchar(c) ;
            END ;
            IF NOT error OR c = ' ' OR c = '.'
            THEN part :- yname.Strip
            ELSE BEGIN
               yname :- yname.Strip ;
               IF yname = "CMS:" OR yname = "TTY:"
                  THEN error_flag := FALSE
                ELSE BEGIN
                     error_flag := TRUE ;
                     Outtext(title) ;
                     Outtext(" invalid character in file id");
                     Outimage ;
                     Outtext(xname) ; Outimage;
                     Outtext(Blanks(xname.Pos -1));
                     Outchar(mark_char);
                     Outimage;
                END ;
               part :- Copy(yname) ;
            END;
         END
         ELSE part:-NOTEXT;
      END of part;
```

```
COMMENT

Generates DDNAMEs in sequence with the following format:
'SIMxxx', where xxx is a three digit number beginning with
100.  Each successive call produces the next in the sequence.
Global integer variable DDCOUNT is required.    ;

TEXT PROCEDURE simxxx ;
BEGIN TEXT t ;
   ddcount := ddcount + 1 ;
   t :- Blanks(10) ;
   t.Putint(ddcount) ;
   t.Sub(5,3) := "SIM" ;
   simxxx :- Copy(t.Sub(5,6));
END of simxxx;
```

```
        COMMENT


        Issues a filedef for the disk file FILE_NAME.
        Inputs are: 1. the ddname to use, 2. file id, 3. recfm,
        4. lrecl, and options if desired.

        Filedef format is:
        FILEDEF ddname DISK file_name (RECFM recfm LRECL length options

        Options should not inclued any of the followins:
        1. lowcase, 2. lrecl, 3. recfm, and 4. perm.

        If the filedef is sucessful a TRUE value is returned otherwise FALSE.  ;

        BOOLEAN PROCEDURE filedef(file_name,ddname,recfm,lrecl,options);
        VALUE file_name,ddname,recfm,lrecl,options ;
        TEXT file_name,ddname,recfm,options ;
        INTEGER lrecl ;
        INSPECT tty DO
        BEGIN
           TEXT ARRAY fd(1:10) ;
           INTEGER l,n,m,retcode ;
           TEXT cmd ;

          IF file_name ^= "TTY:"
             THEN BEGIN
                  fd(1) :- Copy("FILEDEF") ;
                  fd(2) :- ddname ;
                  fd(3) :- Copy("DISK") ;
                  fd(4) :- file_name ;
                  fd(5) :- Copy("(RECFM") ;
                  fd(6) :- recfm ;
                  fd(7) :- Copy("LRECL") ;
                  fd(8) :- Blanks(10) ;
                  fd(8).Putint(lrecl) ;
                  fd(8):-fd(8).Sub(7,4) ;
                  fd(9) :- options ;
             END
             ELSE BEGIN
                  fd(1) :- Copy("FILEDEF") ;
                  fd(2) :- ddname ;
                  fd(3) :- Copy("TERMINAL (LRECL 132 ");
```

```
            fd(4) :- options ;
        END ;

        cmd :- Blanks(80);
        m := 1 ;
       n := 1 ;
       WHILE fd(n) ^= NOTEXT AND n <= 9 DO BEGIN
           l := fd(n).Length ;
           cmd.Sub(m,l) := fd(n) ;
           m := m + l + 1 ;
         n := n + 1 ;
        END ;

        cmscommand(cmd,retcode) ;

        IF retcode ^= 0
        THEN BEGIN
           Outtext("? Problem with file definition for: ") ;
           Outimage ;
           Outtext(cmd) ; Outimage ;
           filedef := FALSE ;
        END ELSE filedef := TRUE ;

    END of filedef;
```

```
        COMMENT

        Prompt the user for an input file id with the prompt supplied
        by the calling program.  Checks the syntax of the response and
        locates the file.

        Using the stack option of LISTFILE, the recfm and lrecl are read
        and used to issue a filedef for the file.  The ddname is selected
        by the procedure SIMXXX and is returned to the calling program
        LENGTH is set to the files lrecl.

        If any problem occure, an error message is printed and the user
        is requested to reenter the file id.

        The file name part of the file id has no default but the
        file type defaults to "DATA" and the file mode to "A".  ;


        TEXT PROCEDURE set_input_file(prompt,lrecl) ;
        NAME lrecl ;
        VALUE prompt ; TEXT prompt ;
        INTEGER lrecl ;

        INSPECT tty DO
        BEGIN

           TEXT ARRAY token(1:10) ;
           TEXT cmd,file_name,xp,title,empty,options,recfm,ddname ;
           INTEGER n,m,l,retcode ;
           BOOLEAN ok,error ;

           empty :- NOTEXT ;
           title :- Copy("? ") ;

          xp :- conc2(prompt," input file? ");

        COMMENT  Request the file id from the user. ;

        rats: ok := FALSE ;
           WHILE NOT ok DO BEGIN
              file_name :- text_request(xp,"CMS:",fdhelp) ;
```

```
        COMMENT  Disassemble the file id and check for syntax.  ;

            ok := TRUE ;
            FOR n := 1 STEP 1 UNTIL 10 DO BEGIN
               token(n) :- part(title,error,file_name) ;
               IF error THEN ok := FALSE ;
            END ;
         END ;

        COMMENT  If the user responsed with CMS: go into CMS_SUBSET   ;

          IF token(1) = "CMS:"
             THEN BEGIN
                  cms_subset ;
                  GO TO rats ;
             END ;

        COMMENT If the response was not TTY:, he wants a disk file instead
                of a Terminal file.          ;

          IF token(1) ^= "TTY:"
             THEN BEGIN

        COMMENT  Set default file type and file mode.
                 Check for an empty file name part.
                 No defaults allowed for file name part.
                                                      ;

            IF token(2) = empty THEN token(2) :- Copy("DATA") ;
            IF token(3) = empty THEN token(3) :- Copy("A") ;

            IF token(1) = empty
            THEN BEGIN
               Outtext("? file name part is missing") ;
               Outimage;
               ok := FALSE ;
            END ;
```

```
        COMMENT  The syntax is ok.  Check to see if the file exist. ;

            IF ok THEN BEGIN
               cmd :- Blanks(80) ;
               m := 11 ;
               FOR n := 1 STEP 1 UNTIL 3 DO BEGIN
                  l := token(n).Length ;
                  cmd.Sub(m,1) := token(n) ;
                  m := m + l + 1 ;
               END ;

               file_name :- cmd.Sub(11,m - 12);
               cmd.Sub(1,5) := "STATE" ;
               cmscommand(cmd,retcode) ;
               IF retcode ^= 0
               THEN BEGIN
                  ok := FALSE ;
                  Outtext("? file not found") ;
                  Outimage;
               END ;
            END ;


        COMMENT If there were any problems so back and try again. ;

            IF NOT ok THEN GO TO rats ;

        COMMENT  Syntax ok and file exist.  Build filedef options form
                 tokens.  ;

            m := 1 ;
            options :- Blanks(80);
            n := 4 ;
            WHILE token(n) ^= empty DO BEGIN
               l := token(n).Length ;
               options.Sub(m,1) := token(n) ;
               m := m + l + 1 ;
               n := n + 1 ;
            END ;
            options :- options.Strip ;
```

```
        COMMENT  Using the stack option of LISTFILE, stack the
                 files RECFM and LRECL on the console stack.
                 If the return code is not 0 the file exist
                 on an extension disk of the 'A' disk.  Set
                 the file mode to '*' and rebuild the file mode.

                                                              ;

           cmd.Sub(1,8) := "LISTFILE" ;
           cmd.Sub(file_name.Length+12,9) := "(FO STACK" ;
           cmscommand(cmd,retcode) ;

           IF retcode ^= 0
           THEN BEGIN
              token(3) :- Copy("*") ;
              m := 11 ;
              FOR n := 1 STEP 1 UNTIL 3 DO BEGIN
                 l := token(n).Length ;
                 cmd.Sub(m,l) := token(n) ;
                 m := m + l + 1 ;
              END ;

              cmscommand(cmd,retcode) ;

           END;


        COMMENT  Read the first record stacked for the RECFM and LRECL.
                 The procedure DESBUF will clean off any junk from the
                 console stack.  ;

           Inimage ;
           cmscommand("DESBUF",retcode) ;
           Sysin.Setpos(22) ;
           recfm :- part(title,error,Sysin.Image) ;
           lrecl := Inint ;

         END
         ELSE BEGIN
              file_name :- token(1) ;
              options :- Copy("LOWCASE") ;
         END of TTY case ;
```

```
   ddname :- simxxx ;

 COMMENT  Issue a filedef for the file.  If there are any
          problems, go back and try again.  ;

   IF NOT filedef(file_name,ddname,recfm,lrecl,options)
   THEN GO TO RATS ;

   set_input_file :- ddname ;

 COMMENT  Return the ddname and set the lrecl via the
          parameter list.  ;


 END of set_input_file;
```

```
COMMENT

Request a file id from the user for an output file using the
prompt from the calling program.  Checks the response for
syntax and sets the default file type and file mode if needed.
Issues a filedef for the file using the ddname selected by
SIMXXX.

If any problems occur, an error message is printed and the
user is requested to reenter the file id.  ;



TEXT PROCEDURE set_output_file(prompt,lrecl) ;
VALUE lrecl ;
VALUE prompt ; TEXT prompt ;
INTEGER lrecl ;

INSPECT tty DO
BEGIN

   TEXT ARRAY token(1:10) ;
   TEXT cmd,file_name,xp,title,empty,options,recfm,ddname ;
   INTEGER n,m,l,retcode ;
   BOOLEAN ok,error ;

   empty :- NOTEXT ;
   title :- Copy("? ") ;

   xp :- conc2(prompt," output file?");

COMMENT  Request the file id form the user.  ;


   rats: ok := FALSE ;
   WHILE NOT ok DO BEGIN
      file_name :- text_request(xp,"CMS:",fdhelp) ;


COMMENT  Tear down the user response and check for syntax.  ;
```

```
        ok := TRUE ;
        FOR n := 1 STEP 1 UNTIL 10 DO BEGIN
           token(n) :- part(title,error,file_name) ;
           IF error THEN ok := FALSE ;
        END ;
     END ;

  COMMENT If the user entered CMS:, enter CMS_SUBSET  ;


    IF token(1) = "CMS:"
       THEN BEGIN
            cms_subset ;
            GO TO rats ;
       END ;

  COMMENT If he entered TTY: then he wants a terminal instead of a file  ;


    IF token(1) ^= "TTY:"
       THEN BEGIN

  COMMENT  Set default file type and mode.  Check for a
           missing file name.  No default for the file
           name part is allowed.   ;


      IF token(2) = empty THEN token(2) :- Copy("LOG") ;
      IF token(3) = empty THEN token(3) :- Copy("A") ;

      IF token(1) = empty
      THEN BEGIN
         Outtext("? file name part is missing") ;
         Outimage ;
         ok := FALSE ;
      END ;

      IF NOT ok THEN GO TO rats ;

  COMMENT  The file id is free of syntax errors.
           build the file id form the first three tokens.  ;
```

```
      file_name :- Blanks(80) ;
      m := 11 ;
      FOR n := 1 STEP 1 UNTIL 3 DO BEGIN
         l := token(n).Length ;
         file_name.Sub(m,l) := token(n) ;
         m := m + l + 1 ;
      END ;


      file_name :- file_name.Strip ;


   COMMENT  Conc the remaining tokens as options. ;


      m := 1 ;
      options :- Blanks(80);
      n := 4 ;
      WHILE token(n) ^= empty DO BEGIN
         l := token(n).Length ;
         options.Sub(M,L) := token(n) ;
         m := m + l + 1 ;
         n := n + 1 ;
      END ;
      options :- options.Strip ;

      recfm :- Copy("F") ;

    END
    ELSE BEGIN
        file_name :- token(1) ;
        options :- Copy("LOWCASE") ;
    END of TTY case ;

     ddname :- simxxx ;

   COMMENT  If the filedef has problem so back and try it all
            over.  ;
     IF NOT filedef(file_name,ddname,recfm,lrecl,options)
```

```
        THEN GO TO RATS ;

        set_output_file :- ddname ;

      COMMENT  Return the ddname to the calling program.    ;

      END of set_output_file;
```

```
        COMMENT

        Filedef help procedure used by GET_DD_INPUT, GET_DD_OUTPUT,
        GET_INPUT_FILE, and GET_OUTPUT_FILE.   ;

        BOOLEAN PROCEDURE fdhelp ;
        INSPECT tty DO
        BEGIN
           TEXT spaces ;
           spaces :- Blanks(5) ;

           Outtext("file id syntax"); Outimage;
            Outtext(spaces);
            Outtext("fn ft fm options");
            Outimage;
           Outtext("fn - file name part (no default)"); Outimage;
           Outtext("ft - file type part (input default: data,");
           Outimage;
           Outtext(Blanks(20));Outtext("output default: log )");
           Outimage;
           Outtext("fm - file mode part (default: a)");
           Outimage;
           Outtext("options - filedef options");
           Outimage;
           Outtext(spaces);
           Outtext("should not include any of the following");
           Outimage;
           Outtext(spaces);Outtext("1.lowcase");Outimage;
           Outtext(spaces); Outtext("2.recfm"); Outimage;
           Outtext(spaces); Outtext("3.lrecl"); Outimage;
           Outtext(spaces);Outtext("4.perm");Outimage;
           Outtext("notes: each item entered ");
           Outtext("may be seperated by any number of");
           Outimage;
           Outtext("spaces or a single '.'.example:");
          Outimage;
           Outtext(spaces);Outtext("foo..b");
          Outimage;
           Outtext("results: fn = foo, ft = default, fm = b");
           Outimage;
           fdhelp := FALSE ;
```

```
END of fdhelp;
```

```
BOOLEAN null_line ;

PROCEDURE tty_inimage(f); REF(Infile) f ;

COMMENT  This replaces an inimage for terminal I/O.
         When the user input a null line which would
         cause an end of file on sysin, this routine
         detects it and closed the file and reopens it.
        This means that an empty line is read as an empty line
        and there is no end of file on the terminal.
        ;


BEGIN
    f.Inimage ;
    null_line := FALSE ;
    IF f.Image.Sub(1,2) = "/*"       COMMENT the eof mark ;
      THEN BEGIN
          next_file(f);
          null_line := TRUE ;
       END
       ELSE f.Image.Setpos(1) ;

END of tty_inimage;
%copy OUTFI
```

```
        ddcount := 99 ;
      mark_char := '^';
      default_char := '/';
      help_char := '?';
      prompt_char := ':';

      initialize_terminal;

END of dialog;
```

APPENDIX E

SOURCE LISTING OF

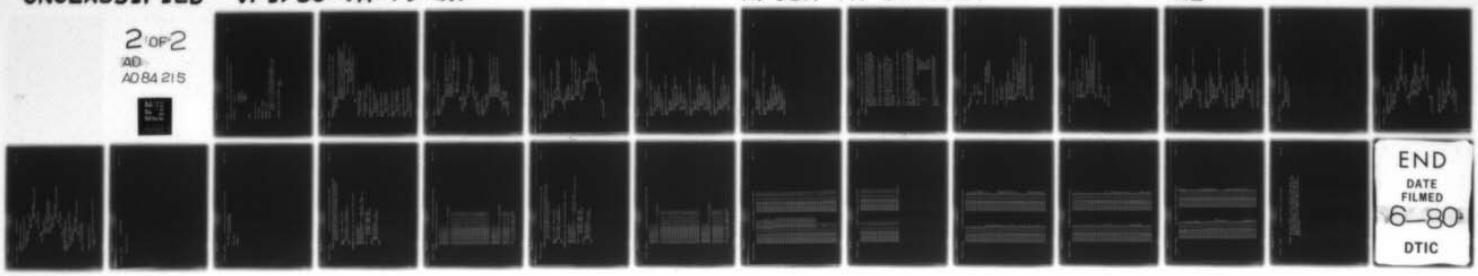
CLASS OUT_FILE

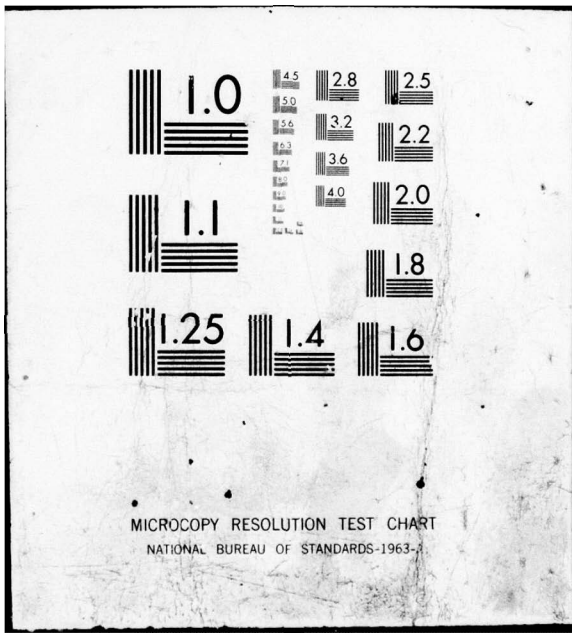
AND OTHER PROCEDURES FROM CLASS DIALOG

```
    COMMENT Copyright 1979 by Richard J. Orgass.

            This information is subject to change without notice.
            The author assumes no responsibility for any
            errors that may be present in this document.
            ;


    COMMENT  The following procedures and arrays are used in the
            apl implementation to translate input EBCDIC characters
            into key-paired ASCII characters and to translate
            key-paired ASCII characters into EBCDIC.  The details
            of these procedures depends on the host machine's
            character set.

            As defined in Section 31.1 of "A Primitive Recursive
            Semantics and Implementation of APL", the function
            kp_code returns the ASCII code of a simple character
            which is its parameter.  In the IBM SIMULA implementation,
            the characters transmitted to the program are EBCDIC
            characters which are translations of the ASCII characters
            typed on the terminal in accord with the translation
            table that appears in Appendix A of TM 79-8.  Therefore,
            the definition of kp_char requires a translation from an
            EBCDIC character to an ASCII character code.  This is
            accomplished using the rank of the EBCDIC character as
            an index in the vector ASCII.

            The definition of kp_char requires that the character that
            is returned is the character in the host machine's character
            set that corresponds to the (real) ASCII code that is its
            parameter.  This is accomplished using the ASCII code to
            index a vector of EBCDIC characters.  This vector is called
            EBCDIC.  It also depends on the translate table in TM79-8.
            ;


        REAL ARRAY ascii(0:255);
        CHARACTER ARRAY ebcdic(0:127);


        CHARACTER PROCEDURE kp_char(n); REAL n;
            kp_char := ebcdic(n);
```

```
REAL PROCEDURE kp_code(c); CHARACTER c;
    kp_code := ascii(Rank(c));
```

AD-A084 215 VIRGINIA POLYTECHNIC INST AND STATE UNIV WASHINGTON --ETC F/G 9/2
DESIGN FOR A CMS SIMULA FILE SYSTEM WITH 4 CHARACTER SETS.(U)
NOV 79 R J ORGASS AFOSR-79-0021
UNCLASSIFIED VPI/SU-TM-79-8A AFOSR-TR-80-0337 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
CLASS out_file(f_spec); VALUE f_spec; TEXT f_spec;

PROTECTED       f_spec,
                f,
                f_opened,
                t_type,
                check_open,
                zyaboi,
                zyamoi;

BEGIN


REF(Outfile) f;

TEXT Image;

BOOLEAN f_opened;

INTEGER t_type;

COMMENT t_type = 0 <=> ASCII terminal or file
        t_type = 1 <=> key-paired APL terminal or file
        t_type = 2 <=> bit-paired APL terminal or file
        ;

EXTERNAL assembly PROCEDURE zyaboi,
                            zyamoi;
```

```
    PROCEDURE check_open(caller); VALUE caller; TEXT caller;
        IF NOT f_opened
            THEN BEGIN
                    tty.Outtext("% out_file.");  tty.Outtext(caller);
                    tty.Outtext(": File '");  tty.Outtext(f_spec);
                    tty.Outtext("' is closed.");  tty.Outimage;
                    IF boolean_request("Do you want to open this file?",
                                       TRUE, TRUE)
                        THEN Open(Blanks(integer_request(
                                  "Enter length of output lines: ",
                                  80, 10, 800,TRUE)))
                        ELSE Error("Closed file termination.")
                END;

    BOOLEAN PROCEDURE opened;
        opened := f_opened;

    TEXT PROCEDURE file_spec;
        file_spec :- Copy(f_spec);

    PROCEDURE Setpos(i); INTEGER i;
    BEGIN
        check_open("Setpos");
        Image.Setpos(i);
    END of Setpos;

    INTEGER PROCEDURE Pos;
    BEGIN
        check_open("Pos");
        Pos := Image.Pos;
    END of Pos;

    BOOLEAN PROCEDURE More;
    BEGIN
        check_open("More");
        More := Image.More;
    END of More;

    INTEGER PROCEDURE Length;
    BEGIN
        check_open("Length");
        Length := Image.Length;
    END of Length;
```

```
PROCEDURE Open(buf); TEXT buf;
BEGIN
    IF f_opened
        THEN BEGIN
                tty.Outtext("? out_file.Open: File '");
                tty.Outtext(f_spec);
                tty.Outtext("' is already open.");
                tty.Outimage;
                IF boolean_request("Do you want to close the file? ",
                                   TRUE, TRUE)
                    THEN Close
                    ELSE Error("File already open.");
             END;
    Image :- buf;
    IF f_spec NE "TTY:"
        THEN BEGIN
                IF f == NONE
                   THEN f :- NEW Outfile(set_dd_output(
                                f_spec,buf.Length));
                f.Open(buf)
             END;
    f_opened := TRUE;
END of Open;

PROCEDURE Close;
BEGIN
    IF NOT f_opened
        THEN BEGIN
                tty.Outtext("% out_file.Close: File '");
                tty.Outtext(f_spec);
                tty.Outtext("' is already closed.");
                tty.Outimage;
                tty.Outtext("Execution continues.");
                tty.Outimage;
             END;
    IF f_spec NE "TTY:"
        THEN f.Close;
    f_opened := FALSE
END of Close;
```

```
PROCEDURE Outtext(t); VALUE t; TEXT t;
BEGIN
    check_open("Outtext");

    IF t.Length + Image.Pos <= Image.Length
     THEN BEGIN
             TEXT temp;
             temp :- Image.Sub(Image.Pos,t.Length);
             Image.Setpos(Image.Pos + t.Length);
             temp := t
          END
     ELSE BEGIN
             INTEGER cnt;
             Breakoutimage;
             cnt := 1;
             WHILE t.Length - cnt >= Image.Length DO
             BEGIN
                 Image :- t.Sub(cnt,Image.Length);
                 Breakoutimage;
                 cnt := cnt + Image.Length
             END;
             IF cnt <= t.Length
                   THEN BEGIN
                           TEXT temp;
                           temp :- Image.Sub(1,
                              (t.Length-cnt)+1);
                           temp := t.Sub(cnt,
                              (t.Length-cnt)+1);
                           Breakoutimage
                        END;
                Outimage
          END
END of Outtext;
```

```
PROCEDURE Outchar(c); CHARACTER c;
BEGIN
    check_open("Outchar");
    IF NOT Image.More
        THEN Breakoutimage;
    Image.Putchar(c);
END of Putchar;

PROCEDURE Outint(i,w); INTEGER i, w;
BEGIN
    TEXT temp;
    INTEGER t;
    check_open("Outint");
    IF Image.Pos + w > Image.Length
        THEN Breakoutimage;
    t := Image.Pos;
    temp :- Image.Sub(t,w);
    temp.Putint(i);
    Image.Setpos(t+w)
END of Outint;

PROCEDURE Outfrac(i,n,w); INTEGER i, n, w;
BEGIN
    TEXT temp;
    INTEGER t;
    check_open("Outfrac");
    IF Image.Pos + w > Image.Length
        THEN Breakoutimage;
    t := Image.Pos;
    temp :- Image.Sub(t,w);
    temp.Putfrac(i,n);
    Image.Setpos(t+w)
END of Outfrac;

PROCEDURE Outreal(r,n,w); REAL r; INTEGER n, w;
BEGIN
    TEXT temp;
    INTEGER t;
    check_open("Outreal");
    IF Image.Pos + w > Image.Length
        THEN Breakoutimage;
    t := Image.Pos;
    temp :- Image.Sub(t,w);
```

```
        temp.Putreal(p,n);
        Image.Setpos(t+w);
    END of Outreal;

    PROCEDURE Outfix(p,m,w); REAL p; INTEGER m, w;
    BEGIN
        TEXT temp;
        INTEGER t;
        check_open("Outfix");
        IF Image.Pos + w > Image.Length
            THEN Breakoutimage;
        t := Image.Pos;
        temp :- Image.Sub(t,w);
        temp.Putfix(p,m);
        Image.Setpos(t + w)
    END of Outfix;
```

```
    TEXT PROCEDURE conv_to_apl(t); VALUE t; TEXT t;

    COMMENT In accord with the value of the variable t_type,
            an EBCDIC text object is translated into a text
            object that will print in approximately the same
            form on ASCII, key-paired or bit-paired ASCII/APL
            terminals.  The translations performed are as
            follows:

            If t_type = 0, then the actual parameter is
            returned as the value of the procedure.

            If t_type = 1, then the actual parameter is
            modified so that it will print on a key-paired
            ASCII/APL terminal in approximately the same
            form as it would on an ASCII terminal.  In
            particular, the following translations are
            performed:

            Upper case letters are mapped into underscored APL
            letters.

            Lower case letters are mapped into APL letters.

            Except as follows, the ASCII graphics are mapped
            into the corresponding graphic in the APL character
            set.  There is no exact correspondence for the
            following ASCII characters and they are mapped into
            another graphic.

            ASCII Graphic              Translated Graphic
                #                            right tack
                %                            diamond
                &                            and sign
                @                            alpha
                ^                            cent sign
                `                            high minus

            If t_type = 2, then the translation described
            above is performed for bit-paired ASCII/APL
            terminals.
            ;
```

```
BEGIN
    INTEGER i,
            j,
            char_code;

    TEXT    new_version,
            temp,
            t1;

    SWITCH tty_type := ascii_terminal,
                       key_paired_terminal,
                       bit_paired_terminal;

    GO TO tty_type(t_type+1);

    ascii_terminal:

        conv_to_apl :- t;
        GO TO return;

    key_paired_terminal:
        new_version :- Blanks(255);
        temp :- t.Strip;
        temp.Setpos(1);
        FOR i := 1 STEP 1 UNTIL temp.Length DO
        BEGIN
            char_code := Rank(temp.Getchar);
            j := new_version.Pos;
            t1 :- new_version.Sub(j,key_paired(char_code).Length);
            t1 := key_paired(char_code);
            new_version.Setpos(j+key_paired(char_code).Length);
        END;

        conv_to_apl :- new_version.Strip;
        GO TO return;
```

```
            bit_paired_terminal:

                new_version :- Blanks(255);
                temp :- t.Strip;
                temp.Setpos(1);
                FOR i := 1 STEP 1 UNTIL temp.Length DO
                BEGIN
                    char_code := Rank(temp.Getchar);
                    j := new_version.Pos;
                    t1 :- new_version.Sub(j,bit_paired(char_code).Length);
                    t1 := bit_paired(char_code);
                    new_version.Setpos(j+bit_paired(char_code).Length)
                END;

                conv_to_apl :- new_version.Strip;
                GO TO return;

            return:

        END of conv_to_apl;
```

```
PROCEDURE set_ascii;
BEGIN
    check_open("set_ascii");
    IF Image.Strip NE NOTEXT
        THEN Outimage;
    IF t_type > 0
        THEN BEGIN
                Outchar(Char(15)); ! Char(15) = <si>;
                Breakoutimage;
             END;
    t_type := 0
END of set_ascii;


PROCEDURE set_key_paired;
BEGIN
    check_open("set_key_paired");
    IF Image.Strip NE NOTEXT
        THEN Outimage;
    IF t_type = 0
        THEN BEGIN
                Outchar(Char(14)); ! Char(14) = <so>;
                Breakoutimage
             END;
    t_type := 1
END of set_key_paired;


PROCEDURE set_bit_paired;
BEGIN
    check_open("set_bit_paired");
    IF Image.Strip NE NOTEXT
        THEN Outimage;
    IF t_type = 0
        THEN BEGIN
                Outchar(Char(15)); ! Char(15) = <si>;
                Breakoutimage
             END;
    t_type := 2;
END of set_bit_paired;


INTEGER PROCEDURE term_type;
```

```
        BEGIN
            check_open("term_type");
            term_type := t_type;
        END of term_type;
```

```
PROCEDURE Outimage;
BEGIN
    check_open("Outimage");
    IF t_type = 0
        THEN BEGIN
                IF f_spec = "TTY:"
                    THEN BEGIN
                            zyamoi(Image);
                            Image :- Blanks(Image.Length)
                         END
                    ELSE f.Outimage
             END
        ELSE BEGIN
                TEXT holder;
                holder :- Image;
                Image :- Blanks(Image.Length);
                Image := conv_to_apl(holder);
                IF f_spec = "TTY:"
                    THEN BEGIN
                            zyamoi(Image);
                            Image :- Blanks(Image.Length)
                         END
                    ELSE f.Outimage
             END
END of Outimage;


PROCEDURE apl_outimage;
BEGIN
    check_open("apl_outimage");
    IF f_spec = "TTY:"
        THEN BEGIN
                zyamoi(Image);
                Image :- Blanks(Image.Length)
             END
        ELSE f.Outimage
END of apl_outimage;
```

```
PROCEDURE Breakoutimage;
BEGIN
    check_open("Breakoutimage");
    IF t_type = 0
        THEN BEGIN
                IF f_spec = "TTY:"
                    THEN BEGIN
                            zyaboi(Image);
                            Image :- Blanks(Image.Length)
                         END
                    ELSE f.Outimage
             END
        ELSE BEGIN
                TEXT holder;
                holder :- Image;
                Image :- Blanks(Image.Length);
                Image := conv_to_apl(holder);
                IF f_spec = "TTY:"
                    THEN BEGIN
                            zyaboi(Image);
                            Image :- Blanks(Image.Length)
                         END
                    ELSE f.Outimage
             END
END of Breakoutimage;
PROCEDURE apl_breakoutimage;
BEGIN
    check_open("apl_breakoutimage");
    IF f_spec = "TTY:"
        THEN BEGIN
                zyaboi(Image);
                Image :- Blanks(Image.Length)
             END
        ELSE f.Outimage
END of apl_breakoutimage;


COMMENT Initialization;

    f_spec :- upcase(frontstrip(f_spec.Strip));
    t_type := 0
```

```
END of out_file;
```

```
        TEXT ARRAY key_paired(0:255),
                   bit_paired(0:255);

        TEXT temp;

        INTEGER i;
```

```
        COMMENT The following code initializes the array key_paired with
                translates ASCII characters into their key-paired APL
                equivalents as defined in the procedure conv_to_apl.
                ;

            FOR i := 0 STEP 1 UNTIL 255 DO
            BEGIN
                key_paired(i) :- Blanks(1);
                key_paired(i).Putchar(Char(i));
            END;

        COMMENT Map upper case into underlined APL letters;

            temp :- Blanks(3);
            temp.Setpos(2);
            temp.Putchar(Char(22)); ! Char(22) = <bs>;
            temp.Putchar('F');      ! underscore in APL set;

            FOR i := 193 STEP 1 UNTIL 201,
                     209 STEP 1 UNTIL 217,
                     226 STEP 1 UNTIL 233 DO
            BEGIN
                temp.Setpos(1);
                temp.Putchar(Char(i-64));
                key_paired(i) :- Copy(temp);
            END;
```

```
        COMMENT Provide translations for graphics;

            temp.Setpos(1);
            temp.Putchar('.');
            temp.Setpos(3);
            temp.Putchar('K');
            key_paired(Rank('!')) :- Copy(temp);

            key_paired(Rank('"')) := "!";
            key_paired(Rank('#')) := "\";
            key_paired(Rank('$')) := "~";
            key_paired(Rank('%')) := "`";
            key_paired(Rank('&')) := ")";
            key_paired(Rank('''')) := "K";
            key_paired(Rank('(')) := ":";
            key_paired(Rank(')')) := """";
            key_paired(Rank('*')) := "P";
            key_paired(Rank('+')) := "-";
            key_paired(Rank('-')) := "_";
            key_paired(Rank(':')) := ">";
            key_paired(Rank(';')) := "<";
            key_paired(Rank('<')) := "#";
            key_paired(Rank('=')) := "%";
            key_paired(Rank('>')) := "&";
            key_paired(Rank('?')) := "Q";
            key_paired(Rank('@')) := "A";
            key_paired(Rank('[')) := ";";
            key_paired(Rank('\')) := "?";
            key_paired(Rank(']')) := "'";

            temp.Setpos(1);
            temp.Putchar('Z');
            temp.Setpos(3);
            temp.Putchar('M');
            key_paired(Rank('^')) :- Copy(temp);

            key_paired(Rank('_')) := "F";
            key_paired(Rank('!')) := "M";
            key_paired(Rank('~')) := "T";
            key_paired(Rank('`')) := "@";

            FOR i := 0 STEP 1 UNTIL 255 DO
                key_paired(i).Setpos(1);
```

```
        COMMENT This code initializes the translation table used
                to translate ASCII into bit-paired APL.  The trans-
                lation is defined in conv_to_apl.
                ;

            FOR i := 0 STEP 1 UNTIL 255 DO
            BEGIN
                bit_paired(i) :- Blanks(1);
                bit_paired(i).Putchar(Char(i));
            END;

        COMMENT Map upper case into underlined APL letters;

            temp :- Blanks(3);
            temp.Setpos(2);
            temp.Putchar(Char(22));    ! Char(22) = <bs>;
            temp.Putchar('F');         ! underscore in APL character set;

            FOR i := 193 STEP 1 UNTIL 201,
                     209 STEP 1 UNTIL 217,
                     226 STEP 1 UNTIL 233 DO
            BEGIN
                temp.Setpos(1);
                temp.Putchar(Char(i-64));
                bit_paired(i) :- Copy(temp);
            END;
```

```
        temp.Setpos(1);
        temp.Putchar('.');
        temp.Setpos(3);
        temp.Putchar('K');
        bit_paired(Rank('!')) :- Copy(temp);

        bit_paired(Rank('"')) := "!";
        bit_paired(Rank('#')) := "[";
        bit_paired(Rank('$')) := "]";
        bit_paired(Rank('%')) := "\";
        bit_paired(Rank('&')) := "_";
        bit_paired(Rank(''')) := "K";
        bit_paired(Rank('(')) := "+";
        bit_paired(Rank(')')) := "*";
        bit_paired(Rank('*')) := "P";
        bit_paired(Rank('+')) := "-";
        bit_paired(Rank('-')) := "=";
        bit_paired(Rank(':')) := ">";
        bit_paired(Rank(';')) := "<";
        bit_paired(Rank('<')) := "#";
        bit_paired(Rank('=')) := "%";
        bit_paired(Rank('>')) := "'";
        bit_paired(Rank('?')) := "Q";
        bit_paired(Rank('@')) := "A";
        bit_paired(Rank('[')) := "$";
        bit_paired(Rank('\')) := "?";
        bit_paired(Rank(']')) := ":";

        temp.Setpos(1);
        temp.Putchar( Z');
        temp.Setpos(3);
        temp.Putchar('M');
        bit_paired(Rank('^')) :- Copy(temp);

        bit_paired(Rank('_')) := "F";
        bit_paired(Rank('!')) := "M";
        bit_paired(Rank('~')) := "T";
        bit_paired(Rank('{')) := "]";
        bit_paired(Rank('}')) := "}";
        bit_paired(Rank('`')) := """";

        FOR i := 0 STEP 1 UNTIL 255 DO
            bit_paired(i).Setpos(1);
```

```
ebcdic(0) := Char(0);
ebcdic(1) := Char(1);
ebcdic(2) := Char(2);
ebcdic(3) := Char(3);
ebcdic(4) := Char(55);
ebcdic(5) := Char(45);
ebcdic(6) := Char(46);
ebcdic(7) := Char(47);
ebcdic(8) := Char(22);
ebcdic(9) := Char(5);
ebcdic(10) := Char(37);
ebcdic(11) := Char(11);
ebcdic(12) := Char(12);
ebcdic(13) := Char(13);
ebcdic(14) := Char(14);
ebcdic(15) := Char(15);
ebcdic(16) := Char(16);
ebcdic(17) := Char(17);
ebcdic(18) := Char(18);
ebcdic(19) := Char(19);
ebcdic(20) := Char(60);
ebcdic(21) := Char(61);
ebcdic(22) := Char(50);
ebcdic(23) := Char(38);
ebcdic(24) := Char(24);
ebcdic(25) := Char(25);
ebcdic(26) := Char(63);
ebcdic(27) := Char(39);
ebcdic(28) := Char(34);
ebcdic(29) := Char(29);
ebcdic(30) := Char(53);
ebcdic(31) := Char(31);
ebcdic(32) := ' ';
ebcdic(33) := '!';
ebcdic(34) := '"';
ebcdic(35) := '#';
ebcdic(36) := '$';
ebcdic(37) := '%';
ebcdic(38) := '&';
ebcdic(39) := ''';
ebcdic(40) := '(';
ebcdic(41) := ')';
ebcdic(42) := '*';
ebcdic(64) := '@';
ebcdic(65) := 'A';
ebcdic(66) := 'B';
ebcdic(67) := 'C';
ebcdic(68) := 'D';
ebcdic(69) := 'E';
ebcdic(70) := 'F';
ebcdic(71) := 'G';
ebcdic(72) := 'H';
ebcdic(73) := 'I';
ebcdic(74) := 'J';
ebcdic(75) := 'K';
ebcdic(76) := 'L';
ebcdic(77) := 'M';
ebcdic(78) := 'N';
ebcdic(79) := 'O';
ebcdic(80) := 'P';
ebcdic(81) := 'Q';
ebcdic(82) := 'R';
ebcdic(83) := 'S';
ebcdic(84) := 'T';
ebcdic(85) := 'U';
ebcdic(86) := 'V';
ebcdic(87) := 'W';
ebcdic(88) := 'X';
ebcdic(89) := 'Y';
ebcdic(90) := 'Z';
ebcdic(91) := '[';
ebcdic(92) := '\';
ebcdic(93) := ']';
ebcdic(94) := '^';
ebcdic(95) := '_';
ebcdic(96) := '`';
ebcdic(97) := 'a';
ebcdic(98) := 'b';
ebcdic(99) := 'c';
ebcdic(100) := 'd';
ebcdic(101) := 'e';
ebcdic(102) := 'f';
ebcdic(103) := 'g';
ebcdic(104) := 'h';
ebcdic(105) := 'i';
ebcdic(106) := 'j';
```

```
ebcdic(43) := '+';          ebcdic(107) := 'k';
ebcdic(44) := ',';          ebcdic(108) := 'l';
ebcdic(45) := '-';          ebcdic(109) := 'm';
ebcdic(46) := '.';          ebcdic(110) := 'n';
ebcdic(47) := '/';          ebcdic(111) := 'o';
ebcdic(48) := '0';          ebcdic(112) := 'p';
ebcdic(49) := '1';          ebcdic(113) := 'q';
ebcdic(50) := '2';          ebcdic(114) := 'r';
ebcdic(51) := '3';          ebcdic(115) := 's';
ebcdic(52) := '4';          ebcdic(116) := 't';
ebcdic(53) := '5';          ebcdic(117) := 'u';
ebcdic(54) := '6';          ebcdic(118) := 'v';
ebcdic(55) := '7';          ebcdic(119) := 'w';
ebcdic(56) := '8';          ebcdic(120) := 'x';
ebcdic(57) := '9';          ebcdic(121) := 'y';
ebcdic(58) := ':';          ebcdic(122) := 'z';
ebcdic(59) := ';';          ebcdic(123) := '{';
ebcdic(60) := '<';          ebcdic(124) := '!';
ebcdic(61) := '=';          ebcdic(125) := '}';
ebcdic(62) := '>';          ebcdic(126) := '~';
ebcdic(63) := '?';          ebcdic(127) := Char(7);
```

```
ascii(0) := 0;
ascii(1) := 1;
ascii(2) := 2;
ascii(3) := 3;
ascii(4) := 20;
ascii(5) := 9;
ascii(6) := 0;
ascii(7) := 127;
ascii(8) := 0;
ascii(9) := 0;
ascii(10) := 0;
ascii(11) := 11;
ascii(12) := 12;
ascii(13) := 13;
ascii(14) := 14;
ascii(15) := 15;
ascii(16) := 16;
ascii(17) := 17;
ascii(18) := 18;
ascii(19) := 19;
ascii(20) := 0;
ascii(21) := 0;
ascii(22) := 8;
ascii(23) := 0;
ascii(24) := 24;
ascii(25) := 25;
ascii(26) := 0;
ascii(27) := 0;
ascii(28) := 0;
ascii(29) := 29;
ascii(30) := 0;
ascii(31) := 31;
ascii(32) := 0;
ascii(33) := 0;
ascii(34) := 28;
ascii(35) := 0;
ascii(36) := 0;
ascii(37) := 10;
ascii(38) := 23;
ascii(39) := 27;
ascii(40) := 0;
ascii(41) := 0;
ascii(42) := 0;
ascii(128) := 0;
ascii(129) := 97;
ascii(130) := 98;
ascii(131) := 99;
ascii(132) := 100;
ascii(133) := 101;
ascii(134) := 102;
ascii(135) := 103;
ascii(136) := 104;
ascii(137) := 105;
ascii(138) := 0;
ascii(139) := 123;
ascii(140) := 0;
ascii(141) := 0;
ascii(142) := 0;
ascii(143) := 0;
ascii(144) := 0;
ascii(145) := 106;
ascii(146) := 107;
ascii(147) := 108;
ascii(148) := 109;
ascii(149) := 110;
ascii(150) := 111;
ascii(151) := 112;
ascii(152) := 113;
ascii(153) := 114;
ascii(154) := 0;
ascii(155) := 125;
ascii(156) := 0;
ascii(157) := 0;
ascii(158) := 0;
ascii(159) := 0;
ascii(160) := 0;
ascii(161) := 126;
ascii(162) := 115;
ascii(163) := 116;
ascii(164) := 117;
ascii(165) := 118;
ascii(166) := 119;
ascii(167) := 120;
ascii(168) := 121;
ascii(169) := 122;
ascii(170) := 0;
```

```
ascii(43) := 0;          ascii(171) := 0;
ascii(44) := 0;          ascii(172) := 0;
ascii(45) := 5;          ascii(173) := 91;
ascii(46) := 6;          ascii(174) := 0;
ascii(47) := 7;          ascii(175) := 0;
ascii(48) := 0;          ascii(176) := 0;
ascii(49) := 0;          ascii(177) := 0;
ascii(50) := 22;         ascii(178) := 0;
ascii(51) := 0;          ascii(179) := 0;
ascii(52) := 0;          ascii(180) := 0;
ascii(53) := 30;         ascii(181) := 0;
ascii(54) := 0;          ascii(182) := 0;
ascii(55) := 4;          ascii(183) := 0;
ascii(56) := 0;          ascii(184) := 0;
ascii(57) := 0;          ascii(185) := 0;
ascii(58) := 0;          ascii(186) := 0;
ascii(59) := 0;          ascii(187) := 0;
ascii(60) := 20;         ascii(188) := 0;
ascii(61) := 21;         ascii(189) := 93;
ascii(62) := 0;          ascii(190) := 0;
ascii(63) := 26;         ascii(191) := 0;
ascii(64) := 32;         ascii(192) := 123;
ascii(65) := 0;          ascii(193) := 65;
ascii(66) := 0;          ascii(194) := 66;
ascii(67) := 0;          ascii(195) := 67;
ascii(68) := 0;          ascii(196) := 68;
ascii(69) := 0;          ascii(197) := 69;
ascii(70) := 0;          ascii(198) := 70;
ascii(71) := 0;          ascii(199) := 71;
ascii(72) := 0;          ascii(200) := 72;
ascii(73) := 0;          ascii(201) := 73;
ascii(74) := 94;         ascii(202) := 0;
ascii(75) := 46;         ascii(203) := 0;
ascii(76) := 60;         ascii(204) := 0;
ascii(77) := 40;         ascii(205) := 0;
ascii(78) := 43;         ascii(206) := 0;
ascii(79) := 124;        ascii(207) := 0;
ascii(80) := 38;         ascii(208) := 125;
ascii(81) := 0;          ascii(209) := 74;
ascii(82) := 0;          ascii(210) := 75;
ascii(83) := 0;          ascii(211) := 76;
ascii(84) := 0;          ascii(212) := 77;
ascii(85) := 0;          ascii(213) := 78;
```

```
ascii(86) := 0;
ascii(87) := 0;
ascii(88) := 0;
ascii(89) := 0;
ascii(90) := 33;
ascii(91) := 36;
ascii(92) := 42;
ascii(93) := 41;
ascii(94) := 59;
ascii(95) := 126;
ascii(96) := 45;
ascii(97) := 47;
ascii(98) := 0;
ascii(99) := 0;
ascii(100) := 0;
ascii(101) := 0;
ascii(102) := 0;
ascii(103) := 0;
ascii(104) := 0;
ascii(105) := 0;
ascii(106) := 124;
ascii(107) := 44;
ascii(108) := 37;
ascii(109) := 95;
ascii(110) := 62;
ascii(111) := 63;
ascii(112) := 0;
ascii(113) := 0;
ascii(114) := 0;
ascii(115) := 0;
ascii(116) := 0;
ascii(117) := 0;
ascii(118) := 0;
ascii(119) := 0;
ascii(120) := 0;
ascii(121) := 96;
ascii(122) := 58;
ascii(123) := 35;
ascii(124) := 64;
ascii(125) := 39;
ascii(126) := 61;
ascii(127) := 34;
ascii(214) := 79;
ascii(215) := 80;
ascii(216) := 81;
ascii(217) := 82;
ascii(218) := 0;
ascii(219) := 0;
ascii(220) := 0;
ascii(221) := 0;
ascii(222) := 0;
ascii(223) := 0;
ascii(224) := 92;
ascii(225) := 0;
ascii(226) := 83;
ascii(227) := 84;
ascii(228) := 85;
ascii(229) := 86;
ascii(230) := 87;
ascii(231) := 88;
ascii(232) := 89;
ascii(233) := 90;
ascii(234) := 0;
ascii(235) := 0;
ascii(236) := 0;
ascii(237) := 0;
ascii(238) := 0;
ascii(239) := 0;
ascii(240) := 48;
ascii(241) := 49;
ascii(242) := 50;
ascii(243) := 51;
ascii(244) := 52;
ascii(245) := 53;
ascii(246) := 54;
ascii(247) := 55;
ascii(248) := 56;
ascii(249) := 57;
ascii(250) := 124;
ascii(251) := 0;
ascii(252) := 0;
ascii(253) := 0;
ascii(254) := 0;
ascii(255) := 0;
```

```
        COMMENT The above EBCDIC to ASCII translate table (ASCII) is
                designed to deal with the characters that can be typed
                on an ASCII terminal.  This implies that characters
                outside the 128 ASCII characters cannot occur and,
                therefore, the remaining EBCDIC codes are mapped into
                ASCII 0, NULL.
                ;
```